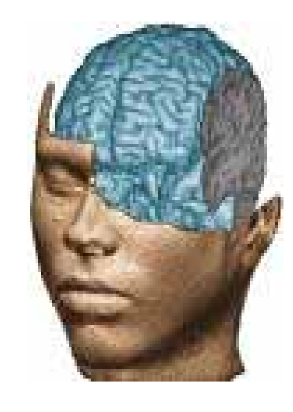
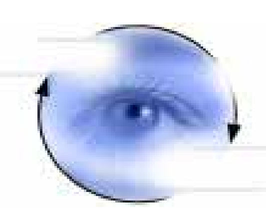
HACK X CRACK: "PORT SCANNING" -- APRENDE A ESCANEAR LA RED





PROTOCOLO HTTP

HYPERTEXT
TRANSFER
PROTOCOL



HACK X CRACK - HACK X CRACK - HACK X CRACK

CURSO







BASES DE DATOS
DISEÑO DE FORMULARIOS Y ORGANIZACION

PROGRAMANDO



VARIABLES Y PUNTEROS

Nº 13 -- P.V.P. 4,5 EUROS



3 SERVIDORES ON LINE PARA TUS PRACTICAS DE HACK

LOS CUADERNOS DE HACK X CRACK
www.hackxcrack.com

ANALIZANDO EQUIPOS REMOTOS

FLAGS

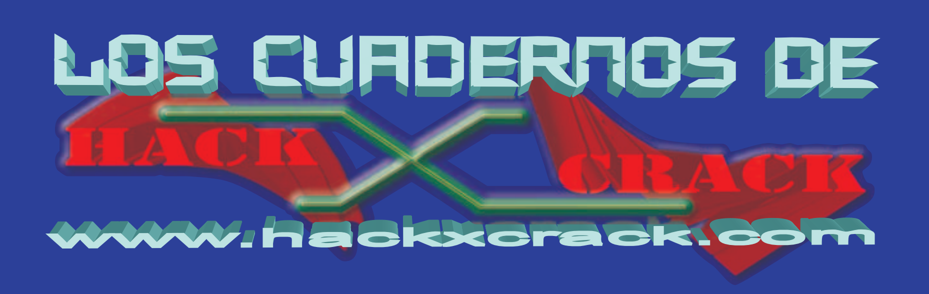
FIN SCAN
ACK SCAN
SYN SCAN
ZOMBIE SCAN
XMAS SCAN
UDP SCAN
NMAP
CONNECT SCAN

ESCANEANDO LA RED

DETECCION REMOTA DE SISTEMAS

PC PASO A PASO: PROTOCOLOS DE INTERNET -- EL VERDADERO CONOCIMIENTO





EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)



Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUSTO, ENTROPIC, MEIDOR, HASHIMUIRA, BACKBONE, ZORTEMIUS, AK22, DORKAN, KMORK, MAILA, TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 80
Petición de Números atrasados y Suscripciones (Srta. Genoveva)

HORARIO DE ATENCIÓN: DE 9:30 A 13:30
(LUNES A VIERNES)

NUMERO 13 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756

DOMINIOS • CORREO PERSONALIZADO • ALOJAMIENTO • CREACION DE PAGINAS • E-COMERCIO • SERVIDORES DEDICADOS



¡Aloje su página a partir de 1 € !/mes

NOMBRE DE DOMINIO

¡NOVEDAD! Opcional: Alojamiento 1 €/mes por cada 10 Mb.

Pack Web Dominio .... **1** €/mes

Su nombre de dominio; .com, .net, .org, .info, .biz... + servicio DNS + 250 subdominios + redireccionamiento web no transparente... Usted es el propietario de su dominio y controla integramente su gestión gracias a nuestras herramientas online (ej.: modificación de DNS, cambio de registrar...).

CORREO PERSONALIZADO

¡NOVEDAD! Opcional: Alojamiento 1 €/mes por cada 10 Mb.

Pack Web Contact .... **2** €/mes

Su nombre de dominio; .com, .net, .org, .info, .biz... + servicio DNS + 250 subdominios + redireccionamiento web transparente + redirección ilimitada de sus correos...

Pack Web Mail .... **3** €/mes

Su nombre de dominio; .com, .net, .org, .info, .biz... + servicio DNS + 10 correos personalizados + alias ilimitados + contestador + redireccionamiento ilimitado de sus correos + redireccionamiento web transparente + webmail + lista de correos + 250 subdominios...

ALOJAMIENTO

Pack Web Pro .... **7,5** €/mes

Su nombre de dominio; .com, .net, .org, .info, .biz..., 10 correos personalizados (ext. a 100), alojamiento de páginas dinámicas 100 Mb. (Ext. a 1 Gb.), tráfico ilimitado, alias ilimitados, webmail, contestador, lista de correos, PHP4, 2 bases de datos MySQL, CGI, Perl 5, FrontPage 2002, acceso FTP privado, estadísticas...

Servidor Privado Linux o Windows .... **19** €/mes

Obtenga todas las ventajas de un servidor dedicado al precio de un servidor compartido. 300 Mb. (ext. a 1,5 Gb.), tráfico ilimitado, 40 aplicaciones preinstaladas (Python, Tomcat,...)…

Pack Web Partner Linux .... **50** €/mes

Webs ilimitadas, correos personalizados ilimitados, alojamiento de páginas dinámicas 1 Gb., PHP4, 50 bases de datos MySQL, CGI, Perl, ofrezca sus propios packs, panel de control avanzado…

UTILIDAD DE CREACIÓN DE PAGINAS ONLINE

¡Pruébalo **Gratis**!

Web Site Creator > **¡¡Alojamiento Gratuito!!** .... a partir de **0,5** €/mes

7 sencillos pasos para construir su propia página web con una calidad profesional. Además con el **Web Site Creator, AMEN** te ofrece el alojamiento gratis o si lo prefieres puedes elegir otro proveedor para el alojamiento. A través de una sencilla interfaz web, nuestra herramienta de creación **Web Site Creator** le permitirá crear, editar y actualizar su página con total autonomía, pudiendo usar las miles de combinaciones posibles.

SERVIDORES DEDICADOS

Estandard: Dirección IP fija, acceso telnet, monitoreo del tráfico, **gratis 512 Kbps extras de ancho de banda en contratos anuales.** Opcional: Reinicio remoto,monitoreo de seguridad (Qualys), certificado SSL, dirección IP suplementaria.

SERVIDOR AMEN 1U .... **149** €/mes **199** €/mes

Procesador Intel Pentium IV 1,8 Ghz., 512 Mb. RAM, 2 Discos duros IDE 40 Gb. 512 Kbps ancho de banda.

COBALT RAQ 550 .... **149** €/mes

Procesador Intel Pentium III 1 Ghz., 256 Mb. RAM, Disco duro 40 Gb. 512 Kbps de ancho de banda.

NUESTROS COMPROMISOS:

- GARANTIA DE REEMBOLSO • SOPORTE TECNICO 7/7 • TRAFICO ILIMITADO • SIN GASTOS DE PUESTA EN MARCHA
- NINGUN GASTO OCULTO • ACTUALIZACION GRATUITA DE UN PACK A OTRO • ADMINISTRACION 100% ONLINE
- DISPONIBILIDAD 99,9% • MONITORIZACION ACTIVA 24/7 • GARANTIA ANCHO DE BANDA REDUNDANTE



902 165 902 **www.amen.es**

EDITORIAL

PREPARANDO EL FUTURO

Cuando uno forma parte de un proyecto, como por ejemplo la publicación de una revista, debe preocuparse tanto de los mil y un problemas diarios como de "el futuro". Hasta hace muy poco, PC PASO A PASO (Los Cuadernos de Hack x Crack), ha sido publicada siguiendo una filosofía tipo "vivir al día", algo que está muy bien en los inicios pero que no debe mantenerse demasiado tiempo.

Ahora estamos preparando toda una serie de cambios cuyos resultados empezarán a "verse" en poco tiempo, desde la Web hasta la revista... nuevos contenidos, publicidad, más páginas, nuevos colaboradores y, sobre todo, mayor diversidad en las temáticas.

Hasta ahora nos hemos centrado en temas relacionados con la Seguridad Informática orientada a Internet y hemos intentado "picar" al lector para que empiece a programar. Ambos temas RED y CURSOS seguirán siendo los principales pilares de PC PASO A PASO, pero estamos trabajando duro para ofrecer mucho más... tiempo al tiempo :)

Para nosotros está siendo muy difícil "madurar", sería sencillo tomar una línea tipo "análisis de productos" y ganar de esta forma publicitantes y lectores tipo "PC ACTUAL", de hecho, hay publicaciones en el mercado cuyas ventas ni siquiera cubren el precio de la Imprenta pero que tienen importantes beneficios por la publicidad. Nosotros estamos trabajando en la cuadratura del círculo, seguiremos con nuestra temática y la ampliaremos, seguiremos siendo educativos en lugar de destructivos, pero empezaremos a diversificar y, sobre todo, aumentar páginas. Esperamos poder demostrar todo este trabajo (que hasta ahora permanece "en la sombra") para los primeros meses del próximo año.

Tratar la temática Hack es tratar la Seguridad Informática, al contrario de lo que muchos piensan ambas cosas son exactamente lo mismo. Algunos creyeron que era imposible hacer una revista de este tipo sin caer en la vulgaridad y sin "empujar" al lector a cometer todo tipo de actos ilícitos, se equivocaron... si algo hemos hecho es enseñar y educar hasta la saciedad.

GRACIAS POR LEERNOS

IDICE DE ANUNCIANTES

IDICE DE ANUNCIANTES

MANIPULACION DE DOCUMENTOS XML: EL DOM

PRIMERA PARTE: TEORIA DEL DOM E INTERFAZ DOMDOCUMENT

POR JOAQUIM ROCA VERGES

XML es una apuesta segura. El común de los mortales lo utilizamos cada día sin saberlo y es un estándar universal implementada en cualquier aplicación que se precie.

Hasta ahora hemos visto lo que es el xml (nº10) y lo que son las DTD (nº 11 y 12). Recordamos que el XML es un lenguaje de intercambio de datos, que hoy en día es prácticamente universal (todo el mundo lo entiende), y las DTD son reglas que podéis aplicar al XML. Ahora vamos a ver como procesar, como recorrer, como extraer la información de los documentos XML.

Y lo vamos a hacer con el DOM.

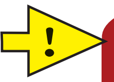
Para daros ánimos e intentar inculcaros mi entusiasmo me gustaría explicaros un par de las muchas experiencias favorables y satisfactorias que he tenido con este lenguaje.

Hace poco estuve diseñando una base de datos bastante grande conjuntamente con otra empresa que llamaremos XXX. Nos reuníamos semanalmente con los usuarios y fruto de las conversaciones con ellos íbamos diseñando la base de datos. El diseño se hacia con Power Designer (podéis bajároslo de http://crm.sybase.com/sybase/www/eBD/my_03/pd952_dwnld_eval.jsp una vez os hayáis registrado), una herramienta de modelado de base de datos de la casa Sybase. La empresa XXX utilizaba Power Designer 7, y mi empresa disponía de Power Designer 8 y 9.

Si la empresa XXX hacia modificaciones y nos las mandaba a mi empresa ningún problema ya que las versiones 8 y 9 entendían a la perfección la versión 7, sin embargo si nosotros hacíamos modificaciones ellos no podían abrir nuestros documentos.

Lo solucionamos guardando nuestras modificaciones como archivo XML, y entonces sí, entonces el colega de la empresa XXX cogía el XML y lo convertía a un archivo Power Designer versión 7.

Hará cosa de un año, fui a una presentación Microsoft de una herramienta BizTalk que sirve para procesar archivos generados con SAP. Los archivos generados con SAP se guardaban... en formato XML.



Para quien se...

Para quien se esté preguntando qué es eso del SAP, le damos otro par de siglas: ERP y CMR. Un ERP es, para que nos entendamos, un programa de gestión empresarial, pero no un "programita" tipo "facturación/nominas", sino todo un sistema de gestión integral, para que te hagas una idea, los bancos y grandes multinacionales utilizan sistemas tipo ERP para gestionar desde los datacenters (centros de datos que pueden llegar a ocupar varias plantas de un edificio e incluso varios edificios) hasta la sucursal de un pequeño pueblo o un simple punto de venta. Alguno estará pensando como es posible "instalar" un ERP (algo tan aparentemente "grande") en un punto de venta (por ejemplo en un pequeño McDonals), pues bien, es posible porque un sistema ERP contiene infinidad de pequeños módulos especialmente adaptados para cada una de las áreas que puede tener una empresa. Pero un ERP es mucho más, porque permite a las empresas ampliar, personalizar e incluso crear módulos completamente nuevos que se adapten perfectamente a cualquier tarea. Pero lo verdaderamente importante no es ni su capacidad ni su modularidad, lo realmente importante es que un ERP permite un análisis global y en tiempo real de cualquier aspecto de una empresa, desde una visión financiera de alto nivel (activos/pasivos globales de toda una multinacional) hasta las ventas en la última hora de una diminuta sucursal de un pequeño pueblo de Murcia.

Pues bien, SAP es uno de los ERP más difundidos. Microsoft, como no podía ser menos, compró una empresa especializada en la creación de Software tipo ERP y comercializa su producto ERP bajo el nombre de AXAPTA-NAVISION. Y nos queda el CMR,

que para no extendernos más diremos que es un importante módulo del Sistema ERP que se encarga de "los contactos", vamos, como el Outlook pero a lo grande ;p (espero que se capte la ironía, comparar Outlook con un CMR es de tan mal gusto como comparar el vinagre con un buen vino :)

No creo que en esta revista se trate nunca en profundidad los sistemas ERP, de hecho, la temática que rodea al mundo de los ERPs es tan basta que daría para hacer una revista solo dedicada a ello. Lo malo es que, como todo lo bueno, apenas vendería un millar de ejemplares… aunque la cosa quizá está cambiando… hasta hace poco únicamente las grandes corporaciones podían acceder a un sistema de este calibre (no solo por el precio, sino por lo complejo de su puesta en marcha, implantación, gestión, mantenimiento…); pero actualmente las empresas que crean este tipo de productos están haciendo un guiño a las medianas e incluso pequeñas empresas sacando al mercado "mini-ERPs" diseñados específicamente para determinados sectores empresariales.

Con esta nota, simplemente pretendemos abrir un poco tus horizontes. Los usuarios de informática muchas veces creemos conocer los programas más importantes/difundidos, pero seguro que muy pocos lectores habrán instalado alguna vez un ERP en su ordenador. Si tenemos en cuenta que los ERP dominan el mundo y que son los programas utilizados por las grandes empresas/instituciones… ¿cómo es posible que la inmensa mayoría de los usuarios no tengan/tengamos ni idea del tema?... Desde aquí te invitamos a que investigues sobre ello ;)

¿QUE ES EL DOM?

DOM = **Document Object Model**, es decir modelo de objetos del documento xml.

Hemos visto que un XML puede ser lo siguiente:

```
<DOCUMENTO>
        <BIENVENIDA> Hola, buenas tardes </BIENVENIDA>
</DOCUMENTO>
```

Pues el modelo de objetos de este documento estaría formado por **TODOS** los elementos que lo componen, o sea por el elemento DOCUMENTO, y por el elemento BIENVENIDA y si por ejemplo el elemento BIENVENIDA tuviera un atributo que se llamara "hora":

```
<DOCUMENTO>
        <BIENVENIDA hora="13:00"> Hola, buenas tardes </BIENVENIDA>
</DOCUMENTO>
```

también sería parte del DOM, y si hubiéramos escrito un comentario:

```
<DOCUMENTO>
        <!-- solo se mostrará a partir de la una del mediodía -->
        <BIENVENIDA hora="13:00"> Hola, buenas tardes </BIENVENIDA>
</DOCUMENTO>
```

este comentario también formaría parte del DOM.

Y (**atención**!!) el texto "Hola, buenas tardes" también es parte del DOM

El DOM es el conjunto de todos los componentes (elementos, atributos, comentarios, entidades, texto...) que conforman el documento XML.

Todos estos componentes se denominan objetos.

Por ejemplo, si nuestro PC fuera un documento XML, el DOM estaría formado por el disco duro, la memoria, el procesador, la pantalla, la frecuencia de la pantalla....

El DOM es independiente del lenguaje de programación con el que estemos trabajando. Podemos acceder al DOM XML tanto con Java, como con Visual Basic, C, JavaScript...

Está basado en lo que se denomina **interfaz de nodos** = un conjunto de métodos (funciones) y propiedades para los objetos (elementos, comentarios, atributos, entidades... del documento XML) DOM.

El DOM es una vista estructurada de un documento XML. Podemos verlo como un árbol, y sus hojas:
Una estructura de tipo árbol y las hojas se llaman nodos.

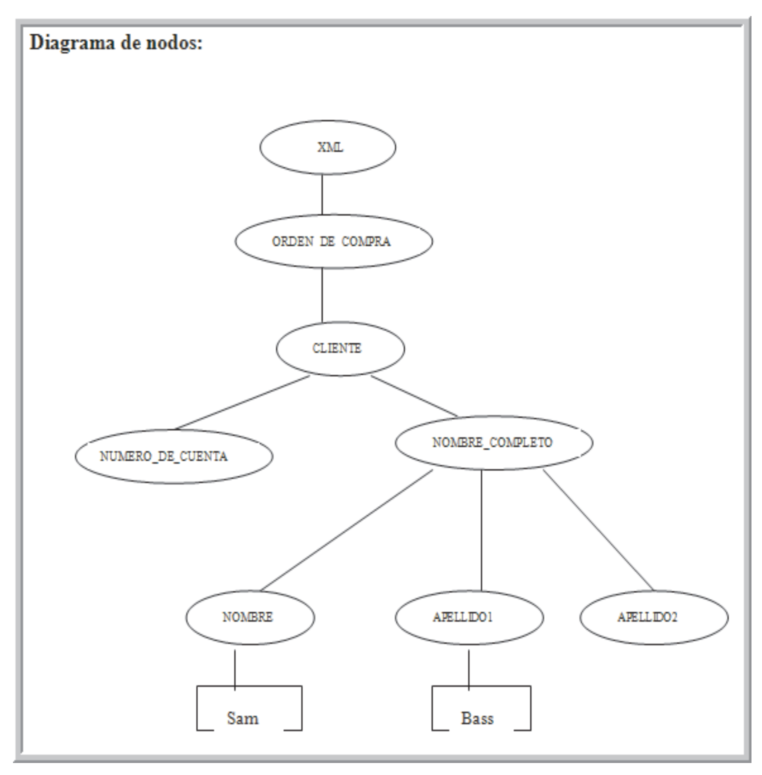
Recordemos el XML de ordenes de compra:

```
<?xml versión="1.0" standalone="yes"?>
<ORDEN_DE_COMPRA>
        <CLIENTE>
<NUMERO_DE_CUENTA>12345678</NUMERO_DE_CUENTA>
                <NOMBRE_COMPLETO>
                        <NOMBRE>Sam</NOMBRE>
                        <APELLIDO1>Bass</APELLIDO1>
                        <APELLIDO2></APELLIDO2>
                </NOMBRE_COMPLETO>
        </CLIENTE>
</ORDEN_DE_COMPRA>
```

Los elementos, comentarios, atributos, entidades, texto etc. que contiene el documento xml , se denominan todos como NODOS, así tenemos que <NOMBRE> es un NODO y "Sam" es otro NODO.
Y el conjunto de NODOS con sus funciones (lo que podemos

hacer sobre los nodos) y propiedades se denomina INTERFAZ DE NODOS.

Y veámoslo representado como un conjunto de nodos que dibujaremos con elipses, y su contenido(el texto) que representaremos con nodos rectángulos (es usual dibujarlo todo en rectángulos, pero he preferido hacer esta primera aproximación de esta manera):



Si esto fuese un objeto Visual Basic, nos gustaría ser capaces de manipular estos nodos, y leer y manipular sus valores. El objeto Visual Basic que contiene el documento xml se llama DocumentObject y los nodos son una matriz de objetos de tipo NODE.

Con lo que la instrucción DocumentObject.Node (index).value nos proporciona el valor de un nodo en particular. Y esto es básicamente lo que el DOM (Document Object Model) XML hace: carga en memoria el documento (con todos sus nodos) y accede, modifica, añade, elimina…

Pero no nos adelantemos.

TIPOS DE NODOS QUE PUEDE CONTENER UN DOCUMENTO XML

Todos los nodos comparten propiedades y métodos comunes, pero dependiendo del tipo de nodo que sea, tiene unas propiedades y métodos particulares.

Tipo de nodo	Valor	Descripción
NODE_ELEMENT	1	El Nodo es un elemento (<ORDEN_DE_COMPRA>)
NODE_ATTRIBUTE	2	El Nodo es el atributo de un elemento (en el ejemplo de BIENVENIDA, sería "horas")
NODE_TEXT	3	El Nodo representa el contenido de un elemento ("Sam", "Bass"), o sea el texto.
NODE_CDATA_SECTION	4	Una sección CDATA. CDATA es una sección en la que podemos escribir cualquier signo, incluidos signos como el mayor que " > " .
NODE_ENTITY_REFERENCE	5	El Nodo es una referencia a una entidad(un video por ejemplo)en el documento xml
NODE_ENTITY	6	El Nodo es la declaración de una entidad
NODE_PROCESSING_INSTRUCTION	7	El Nodo representa una instrucción que hay que procesar
NODE_COMMENT	8	El Nodo representa un comentario en el documento XML
NODE_DOCUMENT	9	El Nodo representa un objeto documento (document object) el cual, como es la raíz del árbol, nos da acceso a TODO el documento XML
NODE_DOCUMENT_TYPE	10	El Nodo representa el DTD
NODE_DOCUMENT_FRAGMENT	11	El Nodo representa un Fragmento de un documento asociado a un nodo que no está contenido en el documento. Lo veis un poco más adelante del artículo con un ejemplo.
NODE_NOTATION	12	El Nodo representa una notación

Cuando estemos programando el DOM, podremos hacer referencia al nodo indicando el tipo de nodo (NODE_ ATTRIBUTE por ejemplo) o a su valor númerico (2 para el NODE_ ATTRIBUTE)

Siguiendo el ejemplo anterior:

Serían NODE_ELEMENT:

<ORDEN_DE_COMPRA>;<CLIENTE>;<NUMERO_DE_CUENTA>;<NOMBRE_COMPLETO>;<NOMBRE>; <APELLIDO1> y <APELLIDO2>

Serían NODE_TEXT: "Sam" Y "Bass"

¿CÓMO MANIPULAMOS EL ARCHIVO XML CON EL DOM?

Para manipular un documento XML, lo primero que tenemos que hacer es cargarlo en la memoria de nuestro ordenador con un parser XML. Nosotros utilizaremos el parser de Microsoft: el Msxml, tal como ya hicimos en el ejemplo de la primera entrega (nº 10).
Una vez el documento está en memoria, ya lo podemos manipular utilizando el DOM

El DOM, trata el documento XML como un árbol. El **DocumentElement** es el elemento superior o raíz del árbol. Este elemento raíz puede tener uno o mas nodos (**nodes**) hijos **(child**) que representarían las hojas del árbol.

Una interfaz, como hemos avanzado antes es un conjunto de métodos (funciones) y propiedades para los objetos (elementos, comentarios, atributos, entidades... del documento XML) DOM.

Para acceder a los objetos del dom (los nodos del DOM) disponemos de cuatro interfaces principales, que trataremos una a una a continuación:

1. **DOMDocument**
2. **XMLDOMNode**
3. **XMLDOMNodeList**
4. **XMLDOMNamedNodeMap**

Para la mayoría de documentos XML, los tipos de nodos más comunes son:

- Elementos
- Atributos
- Texto

Los elementos y el Texto, se tratan con las tres primeras interfaces (DOMDocument, XMLDOMNode y XMLDOMNodeList) Los atributos, no son hijos de ningún otro elemento, sino que forman parte de un elemento. Como caracterizan a un elemento, es decir describen características de un elemento, son diferentes a los otros nodos, no están considerados como hijos de ningún padre y se tratan con la interfaz *XMLDOMNamedNodeMap*

INTERFAZ DOMDocument

Es la interfaz del objeto documento (nodo documento), que incorpora propiedades y métodos para trabajar con el nodo raíz del DOM.
El nodo raíz del DOM representa **TODO** el documento XML y no lo debemos confundirlo con el elemento raíz del documento. Si miráis el diagrama de nodos que hemos dibujado antes, podéis observar el nodo raíz del DOM arriba del todo, con la leyenda XML. Seguidamente y colgando de el podéis ver el elemento raíz del documento que seria el nodo ORDEN_DE_COMPRA.

Vamos a irlo viendo todo con un ejemplo.
Cread en la raíz de vuestro sistema de archivos una carpeta que se llame xmlDom ("C:\xmlDom")

Primero crearemos un DTD, contra el cual se validará nuestro xml. Abrid un editor de texto y escribid:

```
<!ELEMENT ORDEN_DE_COMPRA (CLIENTE)>
    <!ELEMENT CLIENTE (NUMERO_DE_CUENTA,NOMBRE_COMPLETO)>
    <!ELEMENT NUMERO_DE_CUENTA ( #PCDATA)>
    <!ELEMENT NOMBRE_COMPLETO ( NOMBRE, APELLIDO1, APELLIDO2) >
    <!ELEMENT NOMBRE (#PCDATA)>
    <!ELEMENT APELLIDO1 (#PCDATA)>
    <!ELEMENT APELLIDO2 (#PCDATA)>
```

Guardad el archivo como ordenCompra.dtd dentro de la carpeta xmlDom

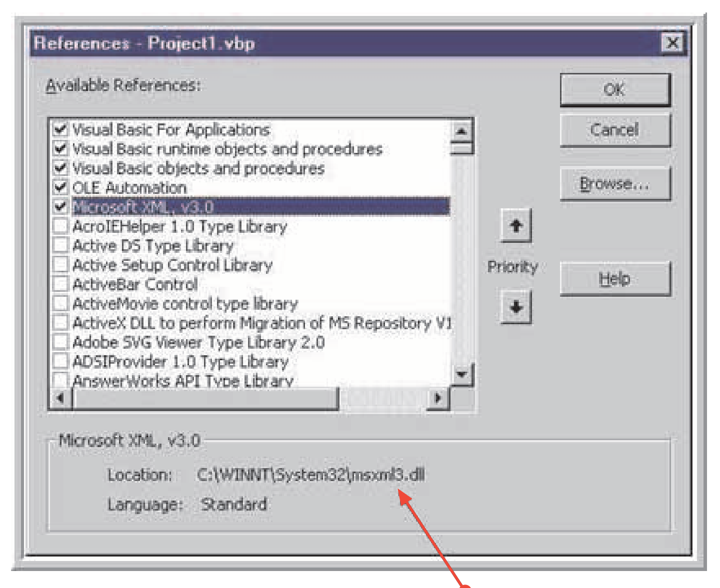
Abrid de nuevo el editor de texto y escribid

```
<?xml version "1.0" standalone="no" ?>
"ordenCompra.dtd">
<ORDEN_DE_COMPRA>
    <CLIENTE>
        <NUMERO_DE_CUENTA>12345678</NUMERO_DE_CUENTA>
        <NOMBRE_COMPLETO>
            <NOMBRE>Sam</NOMBRE>
            <APELLIDO1>Bass</APELLIDO1>
            <APELLIDO2></APELLIDO2>
        </NOMBRE_COMPLETO>
    </CLIENTE>
</ORDEN_DE_COMPRA>
```

y guardad el archivo como Compra.xml

Abrid el Visual Basic y cread un proyecto nuevo tal y como indicamos en el nº 10 (asumimos que tenéis conocimientos básicos de visual basic): esto es

- Abrid el Visual Basic
- Seleccionar un Standard exe
- Añadid una referencia al DOM. Para ello seleccionar del menú PROYECTO la opción REFERENCIAS.
- Buscad una que pone Microsoft XML, v3.0 y seleccionarla, haciendo click en el cuadradito que hay a la izquierda de modo que quede marcado. Y dadle al botón de Aceptar (yo lo tengo en inglés podéis ver que en la pantalla se puede leer OK.)



Fijaros que el nombre de la referencia es a la dll msxml3.

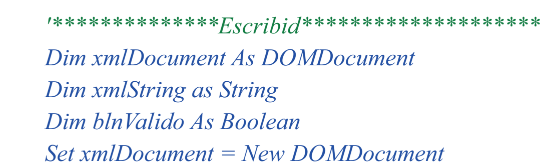
A. DOMDocument CARGAR DEL DOCUMENTO, CREAR UN NUEVO DOCUMENTO XML

Colocad un botón en el formulario, id a las propiedades y poner:

- **name** = cmdCargar
- **caption** = &Cargar documento xml

Codificad el evento click del botón:

- Estas líneas de código lo que hacen **es crear un objeto de tipo DOMDocument:**



```
'*************Escribid******************
Dim xmlDocument As DOMDocument
Dim xmlString as String
Dim blnValido As Boolean
Set xmlDocument = New DOMDocument
'**************************************
```

- Cargar el documento xml en memoria, en el objeto de tipo DOMDocument (método load()), asignarle algunas propiedades y comprobar que está bien formado.

```
'*************Escribid******************
'Antes de cargarlo en memoria, damos valor a alguna de sus propiedades:
'primero le indicamos que se ejecute de manera síncrona, esto es que se ejecute
'inmediatamente, que se cargue inmediatamente
xmlDocument.async = False
'le indicamos que se valide contra el DTD
xmlDocument.validateOnParse = True
'y si hay referencias externas, que las resuelva. Por ejemplo, controlar que el DTD que
'referenciamos existe donde le indicamos, en el path que le indicamos.
xmlDocument.resolveExternals = True
'si el documento es valido, el método Load que es el carga el documento
devolverá un valor true, en caso contrario un valor false
blnValido= xmlDocument.Load ("C:\XmlDom\Compra.xml")
if blnValido then
        MsgBox "El documento es válido"
Else
        MsgBox "El documento no cumple con el DTD"
        Exit sub
End If
'*****************************************************
```



Nota aclaratoria

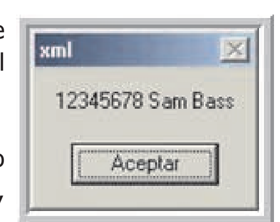
Cuando asignamos un valor a una variable:
Dim i as integer
I= 8
Lo que estamos diciéndole al ordenador es:
Dim i as integer => Resérvame un espacio en memoria para un tipo integer
I= 8 => En el espacio que me has reservado, colócame un 8

En el ejemplo, cuando escribimos
Dim xmlDocument As DOMDocument
Set xmlDocument = New DOMDocument
xmlDocument.Load ("C:\XmlDom\Compra.xml")
Lo que estamos diciendo al ordenador es:
Dim xmlDocument As DOMDocument
Set xmlDocument = New DOMDocument =>resérvame un espacio en memoria para un tipo DomDocument
xmlDocument.Load ("C:\XmlDom\Compra.xml") => ponme en el espacio reservado el archivo Compra.xml

- Pedimos que nos muestre el texto del xml en un mensaje:

```
'**************Escribid*******************
MsgBox xmlDocument.Text
'***************************************
```

Si todo va bien, os mostrará el siguiente message box, con todo el texto del documento:



- Vamos a Crear el documento xml Compras2.xml de manera dinámica, con el método LoadXML

```
'**************Escribid***********
'Ponemos todo el texto del xml en una variable de tipo String
'Fijaros en que he sustituido las dobles comillas de
'version, de standalone y de la ubicación del dtd por 'comillas simples

xmlString = "<?xml version='1.0' standalone='no' ?>" & _
" <!DOCTYPE ORDEN_DE_COMPRA  SYSTEM 'C:\XmlDom\ordenCompra.dtd'> " & _
" <ORDEN_DE_COMPRA> " & _
"    <CLIENTE> " & _
"       <NUMERO_DE_CUENTA>87654321</NUMERO_DE_CUENTA> " & _
"       <NOMBRE_COMPLETO> " & _
"          <NOMBRE>Pat</NOMBRE> " & _
"          <APELLIDO1>Garret</APELLIDO1> " & _
"          <APELLIDO2></APELLIDO2> " & _
"       </NOMBRE_COMPLETO> " & _
"    </CLIENTE> " & _
" </ORDEN_DE_COMPRA>"

'el método loadXML carga en memoria, en el dom, la cadena que le pasamos

xmlDocument.loadXML xmlString
MsgBox xmlDocument.Text

'*************************************************
```



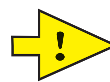
El texto del segundo mensaje será distinto, ahora os saldrá.

- Finalmente, **Guardamos el archivo xml generado**, con un nombre distinto, con el método save

```
'*************Escribid*******************************
'el método save guarda el documento generado o modificado.
xmlDocument.save ("C:\XmlDom\Compra2.xml")
'liberamos memoria
On Error Resume Next
Set xmlDocument = Nothing
'*************************************************
```

B. DOMDocument ACCEDER AL ARBOL DESDE EL ELEMENTO RAIZ, CONTENIDO DE UN NODO EN PARTICULAR

Podemos acceder al árbol del DOM, empezando por la raíz y navegando hacia abajo del árbol (de el nodo mas externo al mas interno) o bien podemos hacer una consulta de un nodo en particular. Navegaremos desde el elemento raíz utilizando la propiedad del domdocument **documentElement** , que nos devuelve el elemento raíz, convertido en un objeto de tipo **XMLDOMNode** (un objeto de la interfaz XMLDOMNode).



Al igual que...

al igual que una propiedad nos puede devolver un valor de tipo booleano, también nos puede devolver un valor de tipo objeto de una clase. En este caso nos devuelve un objeto de la interfaz documentElement. Por eso hemos tenido que declarar una variable de este tipo. Veréis este tipo de asignaciones muy frecuentemente cuando trabajéis con el Dom.

Necesitaremos pues, referenciar de alguna manera la interfaz IXMLDOMElement (el DOM tiene mas interfaces que las cuatro principales; esta se utiliza para nodos de tipo elemento. Es una ampliación de de XMLDomNode) y la interfaz XMLDOMNode. Lo haremos con dos variables de estos tipos. (Mirad el ejemplo)

Para navegar por el documento, vamos a crear un nuevo DTD y un nuevo XML muy parecidos a los que hemos estado utilizando hasta ahora. Vamos a cambiar solamente el nodo raíz, vamos a hacer que ORDEN_DE_COMPRA cuelgue de un nuevo elemento que llamaremos PEDIDOS. Y ampliaremos la información que puede tener ORDEN_DE_COMPRA añadiéndole el elemento PRODUCTO.

Finalmente vamos a indicar a ORDEN_DE_COMPRA que tiene puede tener mas de un orden de compra con el **signo** +.

Abrid un editor de texto y escribid:

```
<!ELEMENT PEDIDOS (ORDEN_DE_COMPRA+)>
        <!ELEMENT ORDEN_DE_COMPRA (CLIENTE, PRODUCTO)>
        <!ELEMENT CLIENTE (NUMERO_DE_CUENTA,NOMBRE_COMPLETO)>
        <!ELEMENT PRODUCTO (#PCDATA)>
        <!ELEMENT NUMERO_DE_CUENTA ( #PCDATA)>
        <!ELEMENT NOMBRE_COMPLETO ( NOMBRE, APELLIDO1, APELLIDO2) >
        <!ELEMENT NOMBRE (#PCDATA)>
        <!ELEMENT APELLIDO1 (#PCDATA)>
        <!ELEMENT APELLIDO2 (#PCDATA)>
```

Guardad el archivo como pedidos.dtd
Abrid de nuevo el editor de texto y escribid:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE PEDIDOS  SYSTEM
        "pedidos.dtd">
<PEDIDOS>
        <ORDEN_DE_COMPRA>
        <CLIENTE>

<NUMERO_DE_CUENTA>12345678</NUMERO_DE_CUENTA>
                <NOMBRE_COMPLETO>
                        <NOMBRE>Sam</NOMBRE>
                        <APELLIDO1>Bass</APELLIDO1>
                        <APELLIDO2></APELLIDO2>
                </NOMBRE_COMPLETO>
        </CLIENTE>
        <PRODUCTO>LIBRO</PRODUCTO>
        </ORDEN_DE_COMPRA>
        <ORDEN_DE_COMPRA>
        <CLIENTE>

<NUMERO_DE_CUENTA>987654321</NUMERO_DE_CUENTA>
                <NOMBRE_COMPLETO>
                        <NOMBRE>Jesse</NOMBRE>
                        <APELLIDO1>James</APELLIDO1>
                        <APELLIDO2></APELLIDO2>
                </NOMBRE_COMPLETO>
        </CLIENTE>
        <PRODUCTO>DISCO DE VINILO</PRODUCTO>
        </ORDEN_DE_COMPRA>
</PEDIDOS>
```

Volved al Visual Basic
Colocad un botón en el formulario, id a las propiedades y poner:

- **name** = cmdArbol
- **caption** = &árbol del documento xml

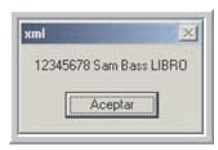


Codificad el evento click del botón:

```
Private Sub cmdArbolDoc_Click()
Dim xmlDocument As DOMDocument
Dim raizDocumento As IXMLDOMElement
Dim nodohijo As IXMLDOMNode
Dim xmlString As String
Set xmlDocument = New DOMDocument
'síncrono, externas y DTD
xmlDocument.async = False
'le indicamos que se valide contra el DTD
xmlDocument.validateOnParse = True
'y si hay referencias externas, que las resuelva.
xmlDocument.resolveExternals = True
If xmlDocument.Load("C:\xmlDom\pedidos.xml") Then
   MsgBox "El documento es válido"
Else
   MsgBox "El documento no cumple con el DTD"
   Exit Sub
End If
'le decimos que raizDocumento = PEDIDOS
Set raizDocumento = xmlDocument.documentElement
'Navegamos por los nodos hijos del elemento raíz
For Each nodohijo In raizDocumento.childNodes
   MsgBox nodohijo.Text
Next
'liberamos la memoria
On Error Resume Next
Set nodohijo = Nothing
Set raizDocumento = Nothing
Set xmlDocument = Nothing

End Sub
```

Fijaros en los dos mensajes que os han salido por pantalla:



Seguido de



Vamos a interpretarlo.

El DOM, ha leído el primer nodo que contenía el nodo raíz. El nodo raíz es PEDIDOS, y el texto que ha leído el DOM es el texto que tenia su primer nodo hijo ORDEN_DE_COMPRA, y el texto que tenia su primer nodo hijo era TODO el texto que contenía ORDEN_DE_COMPRA.

¿Y cual es todo el texto que tiene el **primer nodo** ORDEN_DE_COMPRA?, pues nada más y nada menos

que el texto que tienen todos sus nodos hijos:

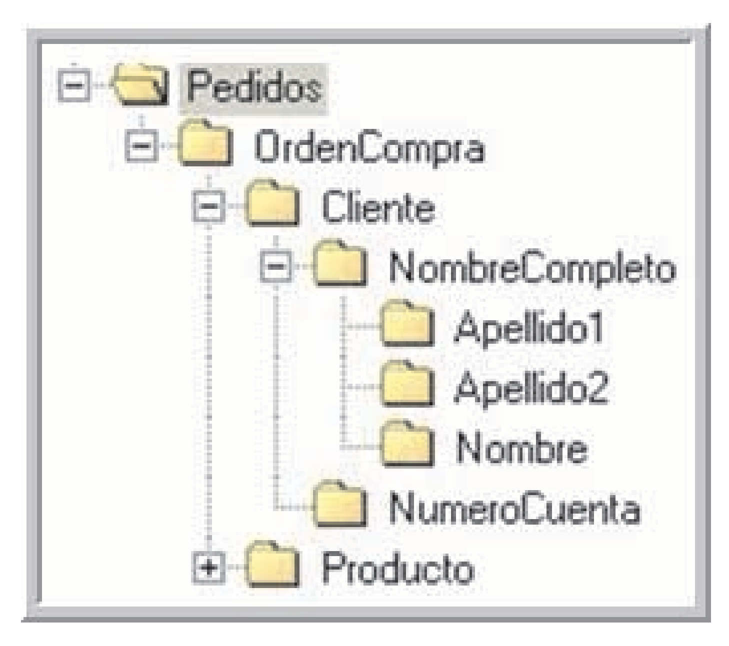
- 12345678
- Sam
- Bass
- LIBRO

Y para el **segundo nod**o ORDEN_DE_COMPRA lo mismo, todo el texto que tienen todos sus nodos hijos:

- 987654321
- Jesse
- James
- DISCO DE VINILO

Para verlo mas claro, pensemos en el explorador de windows.

Supongamos esta estructura de directorios:



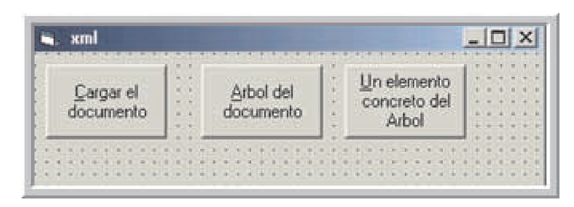
Supongamos que ordenCompra tiene un archivo que se llama 12345678.txt
Supongamos que Apellido1 tiene un archivo que se llama Sam.txt
Supongamos que Apellido2 tiene un archivo que se llama Bass.txt
Supongamos que Producto tiene un archivo que se llama LIBRO.txt

Y pedimos al Sistema Operativo que nos liste todos los archivos que contiene la carpeta Pedidos.... Efectivamente nos devolverá lo mismo que nos ha devuelto el DOM de XML.
Pararos un segundo en pensar y analizar este código.

Si lo entendéis, si os ha entrado bien en la cabeza, lo tenéis muy pero que muy bien.
Para **consultar un nodo en particular**, podemos utilizar el método **getElementsByTagName** que devuelve todos los nodos elemento de un tag en concreto. Devuelve un array que contiene todos los elementos del documento que tienen ese nombre. Recordad que un tag es otra manera de llamar a un elemento: Al elemento <ORDEN_DE_COMPRA> también podéis llámalo tag <ORDEN_DE_COMPRA>

Volved al Visual Basic
Colocad un botón en el formulario, id a las propiedades y poner:

- **name** = cmdUnElemento
- **caption** = &Un elemento concreto del árbol



Codificad el evento click del botón:

```
Private Sub cmdUnElemento_Click()
Dim ListaDeNodos As IXMLDOMNodeList
Dim xmlDocument As New DOMDocument
Dim i As Integer
xmlDocument.async = False
xmlDocument.validateOnParse = True
xmlDocument.resolveExternals = True
If xmlDocument.Load("C:\xmlDom\pedidos.xml") Then
    MsgBox "El documento es válido"
Else
    MsgBox "El documento no cumple con el DTD"
    Exit Sub
End If
MsgBox xmlDocument.xml
Set ListaDeNodos = xmlDocument.getElementsByTagName("NOMBRE")
For i = 0 To (ListaDeNodos.length - 1)
   MsgBox ListaDeNodos.Item(i).xml
   MsgBox ListaDeNodos.Item(i).Text
Next
On Error Resume Next
Set ListaDeNodos = Nothing
Set xmlDocument = Nothing
End Sub
```

Utilizamos la interfaz **IXMLDOMNodeList**, interfaz que nos permite trabajar con la lista de nodos del DOM, tal como su nombre indica.

Fijaros en que el atributo xmlDocument.**xml** os devuelve el contenido de todo el xml, y el atributo del ListaDeNodos.Item(i).**xml** nos devuelve también el contenido de todo el xml que contiene ese nodo, y así como el atributo xmlDocument.**text** nos devuelve el contenido de texto de todo el documento xml, el atributo de la interfaz ixmldomnodelist ListaDeNodos.Item(i).**text** también nos devuelve el texto del nodo que hemos seleccionado. Con ello os quiero demostrar que aunque son interfaces diferentes, el comportamiento es el mismo, si encontráis en alguna otra interfaz las propiedades xml, o text, sin hacer ninguna prueba ya podéis saber lo que devuelve.

C. DOMDocument CREAR UN FRAGMENTO DE XML

Hemos visto hasta ahora las principales propiedades de la interfaz DomDocument, pero no todas. Las otras propiedades son informativas del documento:

- **Doctype**= devuelve el tipo de documento asociado al xml, en nuestro ejemplo nos devolvería el dtd.
- **PreserveWhiteSpace** = que si ponemos a true, preserva los espacios en blanco del documento
- **ReadyState** = nos indica el estado en que se encuentra el documento en este momento. Por ejemplo el documento puede hallarse en estado LOADING (cargando) o en estado COMPLETE (carga completa y el DOM disponible)
- **url** = devuelve una cadena (string) que nos informa de la url de Internet donde se halla nuestro xml

Los métodos mas importantes de la interfaz DomDocument, son los que hemos visto (**load, loadXML, getElementsbyTagName y Save**) y los de creación de **fragmento de xml**. Supongamos que queremos añadir otro ORDEN_DE_COMPRA a nuestro xml: el modo de hacerlo es creando un nuevo nodo ORDEN_DE_COMPRA de acuerdo con el DTD(con todos sus nodos hijos tal como se especifica en el DTD) y una vez creado lo insertaremos en el documento xml.
El tiempo en que este nuevo nodo esta desvinculado de nuestro documento XML, tiempo en el que le estamos añadiendo sus nodos hijos es el tiempo de creación de un **fragmento xml**, que es un trozo de xml a anexar al xml principal.

Todo nuevo nodo se construye con los métodos de la interfaz DOMdocument que comienzan con la palabra **create+tipodenodo**. Por ejemplo para crear un nodo de tipo elemento haremos **createElement**, y para crear un atributo haremos **createAttribute etc.**

Listado de métodos create que se utilizan para crear un fragmento xml:

- **CreateAttribute**
- **CreateCDATASection**
- **CreateComment**
- **CreateDocumentFragment**
- **CreateElement**
- **CreateEntityReference**
- **CreateNode**
- **CreateProcessingInstruction**
- **CreateTextNode**

también podemos, sustituir, insertar los nodos con los métodos

- **InsertBefore**
- **RemoveChild**
- **ReplaceChild**

Veremos un ejemplo de cómo crear un nuevo nodo, en el próximo capítulo, cuando tratemos la interfaz **XMLDOMNode** ya que es necesaria para "enganchar" un nodo hijo a su padre, por ejemplo <NOMBRE_COMPLETO> a <CLIENTE>, o para borrar un nodo hijo, necesitaremos de muchas de las propiedades y métodos que tiene esta interfaz

Tenéis la lista completa de propiedades y métodos de DOMDocument en la dirección:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk30/htm/xmobjxmldomdocument.asp

¡Saludos compañeros!

PROGRAMACION EN GNU LINUX
DESARROLLO DE APLICACIONES EN ENTORNOS UNIX E INICIACIAON AL LENGUAJE C (II)

el_chaman. LUIS U. RODRIGUEZ PANIAGUA

Este mes, en nuestra entrega habitual de LINUX entraremos en el temido mundo de los punteros en C (entre otras cosas). Coged aire y ADELANTE!!!

1. Introducción.

En el artículo anterior vimos como a la hora de realizar un proyecto en un entorno UNiX se suele emplear una modularización del mismo para mejorar la productividad. Así mismo comenzamos una breve introducción al lenguaje C que enlazaba con las estructuras de control de dicho lenguaje vistas en el primer artículo de esta serie.

Este artículo, tal vez por la necesidad de la temática, está orientado a la práctica. Se propondrán algunos ejercicios y búsqueda de información, que espero hagan de su lectura y estudio un tema ameno.

Continuando con lo visto, hoy seguiremos viendo algunos de los tipos disponibles en C. En el número anterior se quedaron en el tintero los tipos derivados debido a que he considerado que el tema de los punteros, vectores y matrices, merece un tratamiento más profundo que el de los tipos básicos. Y sin entretenernos más pasemos ya a los tipos derivados.

2. Tipos Derivados.

Los tipos derivados serán aquellos que se generen a partir de los tipos fundamentales o de otros tipos derivados. Nosotros veremos los arrays o vectores, punteros, estructuras, uniones y campos de bits.

2.1. Arrays o vectores

A casi todos nos sonará el término vector de haberlo visto en matemáticas. Allí considerábamos un vector o arreglo como una colección de bloques de datos. En la programación sucederá algo similar: Un array o vector será una colección de datos del mismo tipo.
Además de esta definición, surgen dos términos asociados al término vector: la dimensión y el índice
Para entender qué es un vector y cómo funcionan estos dos nuevos conceptos, vamos a imaginarnos lo siguiente:

Tal como vimos en el anterior número podemos definir las variables como cajas en las que meter datos. Así mismo vimos que los tipos de dichas variables nos decían de qué tamaño eran dichas cajas.

Dicho esto podremos definir un vector como un conjunto de cajas del mismo tamaño colocadas en fila. El índice será la posición de una determinada caja dentro de estas cajas y la dimensión será cómo podemos colocar en el espacio las distintas filas, columnas, etc... que formarán la geometría de este conjunto de cajas.

```
/* Ejemplo de declaración de varios tipos de vectores o arrays */
/* Vector unidimensional de números enteros */
int   un_vector[5];
/* Vector bidimensional o matriz de números reales  */
float una_matriz[5][5];
/* Vector tridimensional de caracteres */
char  un_cubo[5][5][5];
```

Como podemos observar en el ejemplo, la declaración de un vector corresponde al siguiente esquema:

tipo_array nombre_array[dim1][dim2]..[dimN]

Donde tipo_array será el tipo de datos que contiene el array (es decir el tamaño de cada una de las cajas que lo componen), nombre_array será el nombre que demos al array y [dimX] será el tamaño tope de cada una de las dimensiones, o dicho de otro modo: el número máximo de cajas que se admite en una dimensión.

Para entender mejor el concepto de dimensión, representemos gráficamente el código arriba visto:

Observando el gráfico arriba expuesto, accederemos al contenido de cada casilla mediante sus coordenadas. Obsérvese algo muy importante: En C, siempre los índices empiezan a contar desde 0. Esto quiere decir que si declaramos un vector con cinco casillas, a la hora de poner su dimensión pondremos int un_vector[5] pero a la hora de acceder a todas sus casillas pondremos un_vector[0], un_vector[1], un_vector[2], un_vector[3] y un_vector[4] siendo el número total de casillas cinco, pero siendo el valor máximo del índice una unidad menor que la dimensión total del vector.

Así por ejemplo, para introducir datos en una casilla teclearemos algo como:

```
un_vector[0] = 34;
un_vector[1] = 2 + 3;
una_matriz[2][3] = 3.1416 * 34;
un_cubo[0][2][1] = 'a';
```

Para acceder al contenido se hará de manera similar:

```
int suma;
char respuesta;
...
suma = un_vector[0] + un_vector[1];
printf("%f", una_matriz[2][3]);
...
printf(" Teclee una letra: ");
scanf("%c", &respuesta);
if( un_cubo[0][2][1]==respuesta)
printf("Ha pulsado una a");
```

No se intente...

No se intente buscar sentido alguno a los dos últimos ejemplos: Están hechos con la intención de mostrar el acceso a los componentes de un array. Fíjense en cómo se acceden a los datos, no en lo que se hace con ellos. Este será un tema que trataremos a lo largo del curso. En el primer ejemplo mostramos como introducir datos en casillas concretas de los distintos arrays dependiendo del tipo de estos, y en el segundo caso mostramos como sacar datos de dichos arrays.

A continuación vamos a hacer un programa que cree una matriz de dimensión 3x4 (tres filas y cuatro columnas) y la llene de con sus coordenadas de la siguiente manera: El número almacenado en cada casilla mostrará en su parte entera el número de fila, y en su parte decimal en número de columna. Así las segunda casilla de la primera fila contendrá un 0.1 (Recordemos que las coordenadas empiezan a contar en C siempre desde 0).

```
/*
 * Programa: ej00.c
 *
 * Descripción: Crea una matriz de dimensión 3x4, llena cada casilla
 * con sus coordenadas de la siguiente manera:
```

```
 *    _____ _____ _____ _____
 *  |_0.0_|_0.1_|_0.2_|_0.3_|
 *  |_1.0_|_1.1_|_1.2_|_1.3_|
 *  |_2.0_|_2.1_|_2.2_|_2.3_|
 *
 * e imprime el contenido de la matriz por pantalla.
 *
 * Compilación:
 *     gcc ej00.c -o ej00
 */
#include <stdio.h>
main()
{
    /* Declaración de la matriz de tres filas por cuatro columnas */
    /* Tiene que ser de números reales */
    float matriz[3][4];
    /* Declaración de variables auxiliares contadoras */
    int i,j;
    /* Llenamos la matriz con datos */
    /* i contará filas */
    for(i=0;i<3;i++)
    {
        /* j contará columnas dentro de cada fila */
        for(j=0;j<4;j++)
        {
            matriz[i][j]= (float)(i + (j * 0.1));
        }
    }
    /* Imprimimos la matriz */
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            printf(" %1.1f ",matriz[i][j]);
        }
        /* Después de imprimir una fila, debemos de saltar una línea*/
        printf("\n");
    }
}
```

En este programa de ejemplo, debemos de abundar en una línea un tanto extraña:

```
matriz[i][j]= (float)(i + (j * 0.1));
```

En un principio puede parecer que efectivamente estamos empleando la fórmula adecuada para representar las coordenadas de la casilla en un único número. Puesto de otra forma:

```
matriz[num_fila][num_col]= num_fila + (num_col * 0.1);
```

Así para la casilla matriz[0][2] tendremos que su contenido será 0 + (0.2) = 0.2.

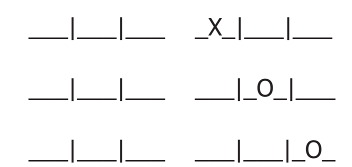
Hasta ahí todos de acuerdo. Ahora bien: ¿ Qué significa eso de poner (float) delante de las variables contadoras i y j?.

A esto se le denomina casting y consiste en transformar el tipo del resultado que nos devuelve una operación cuyos operadores son den tipo distinto a quien va a recibir el dato. Dicho de otra manera: Sirve para cambiar el tipo de un dato en tiempo de ejecución.

Si prestamos atención a la línea de la que estamos hablando, veremos que los operadores son de tipo entero, pero no así el resultado que esperamos conseguir tras la operación. Es por ello que ese resultado lo tenemos que convertir a real. En próximos ejemplos veremos más usos del casting.

2.1.1. Ejercicio 0.0

Se desea hacer un juego del tres en raya para dos jugadores humanos muy sencillo. Las posibles pantallas a mostrar a lo largo del juego pueden ser las siguientes:



Se pide además que el estado del tablero actual se conserve en una matriz de dimensión 3x3 de números enteros.

Entrada y Salida

La entrada y la salida en C se realizan habitualmente sobre los dispositivos de entrada estándar (stdin) y salida estándar (stdout). Tradicionalmente se asocia la entrada estándar al teclado y la salida estándar a la pantalla de la consola, pero esto no tiene porqué ser así siempre. Ya trataremos con ficheros y veremos que stdin y stdout no son más que dos de los muchos flujos de datos que podremos

emplear en un programa.

A efectos prácticos nosotros consideraremos por ahora que la entrada estándar (stdin) es el teclado y tiene una función de lectura asociada llamada scanf y que la salida estándar (stdout) es la consola del ordenador y tiene asociada la función printf.

En conclusión: La lectura de datos del teclado se hará mediante scanf y la impresión de datos en la pantalla de la consola se hará mediante printf.

Un ejemplo de lectura de datos puede ser:

```
int coor_x, coor_y;
...
printf("\n Deme la coordenada X: ")
scanf("%i",&coor_x);
printf("\n Deme la coordenada Y: ");
scanf("%i",&coor_y);
...
```

Observese el & que ponemos delante de la variable donde queremos depositar el dato leído por teclado. Al término del artículo de hoy seremos capaces de comprender qué significa ese & y por qué lo ponemos ahí. Por ahora baste decir que estamos depositando el dato leído en la posición de memoria donde se encuentra la variable donde queremos que se guarde dicho dato.

Un ejemplo de impresión de datos puede ser:

```
int   var_entera=3;
float var_real=3.14;
char  cadena[8]="Hola\0";
char  var_char='a';
printf("Valores:  %i, %f, %s, %c. ", var_entera, var_real, cadena, var_char);
```

A la hora de imprimir el contenido de una variable, (o de leerlo, observar el ejemplo anterior de scanf, tenemos que decir de alguna manera a las funciones de lectura o escritura de datos por entrada y salida estándar respectivamente (scanf y printf) qué es lo que van a leer o escribir. Para ello se emplean las cadenas de formato. Las cadenas de formato son esas letras que comienzan por % y que en función de su valor, indican el tipo de dato con el que estamos tratando :

Tabla 1. Formatos de tipos de dato

Formato	Dato que representa
%i o %d	Entero con signo
%u	Entero sin signo
%f	Número real
%e o %E	Número real, notación exponencial
%g o %G	El compilador decidirá si usa %f o %e (resp. %E)
%c	Un único carácter
%s	Una cadena de caracteres.

Para estos formatos podemos también especificar el ancho de caracteres reservados para expresar un número. Asi %5i reservaría 5 caracteres para imprimir un número entero, independientemente de la longitud de este. %5.2f reservaría 5 caracteres para imprimir la parte entera y dos para la parte decimal, etc...

Obsérvese también, que debemos de poner la cadena de formato en el lugar exacto donde queremos que se imprima el dato, y, posteriormente una lista de variables que irán en el mismo orden el que queremos que aparezca el dato.

Dadas las siguientes variables:

```
int   varA=3;
float varB=4;
char  varC[6]="Hola\0";
char  varD='e';
```

Serán ejemplos correctos:

```
printf(" %3i ", varA);
scanf("%f",&varB);
printf("Una cadena: %s y un real: %3.6f ",varC, varB);
printf("Una letra %c y un número: %i", varD, varA);
printf("Una letra %c y otra letra %c", varD, varC[1]);
printf("Decimal: %i, Octal: %o, Hexadecimal: %x ", varA, varA, varA);
```

E incorrectos:

printf(" %i ", varC);
scanf("%f",&varA);
printf("Una cadena: %s y un real: %f ",varD, varA);
printf("Una letra %c y un número: %i", varA, varD);
printf("Una letra %c y otra letra %c", varD, varC);
printf("Decimal: %i, Octal: %o, Hexadecimal: %x ", varA)

Además de las cadenas de formato, printf admite una serie de caracteres no imprimibles o secuencias de estape como son:

Tabla 2. Secuencias de escape para printf

Secuencia	Descripción
\a	Sonido audible
\b	Retroceso (*backspace*)
\f	Salto de página
\n	salto de línea
\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Imprime \
\'	Imprime '
\"	Sonido audible
\?	Imprime ?
\O	O = cadena de hasta tres dígitos octales
\xH	H = cadena de dígitos hexadecimales

El programa debe se capaz de verificar cuando se ha hecho tres en línea y cuándo ha habido tablas. Así mismo debe de verificar que las coordenadas están dentro del rango permitido y el turno de cada jugador.

2.2. Punteros

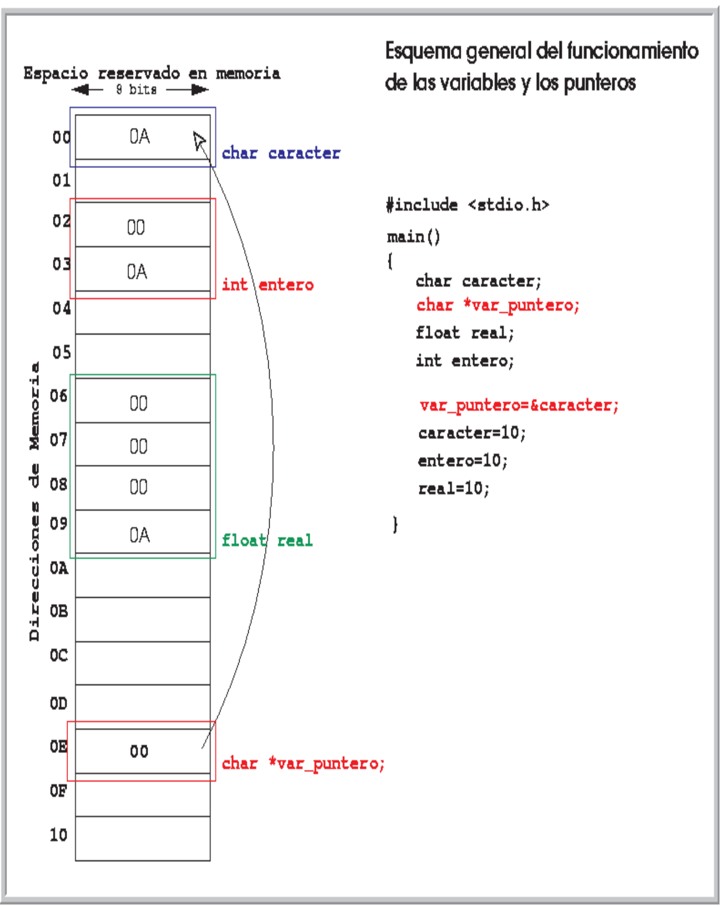
Los punteros suelen ser el caballo de batalla al que se enfrentan los programadores que se adentran por primera vez en lenguajes como C o Pascal. Prescindiendo de esa oscura mitología y de historias espeluznantes que se cuentan relacionadas con los mismos, trataremos de presentar los punteros de una manera clara y asequible, de manera que les perdamos el miedo pero nunca el respeto, dado que son una de las herramientas más potentes que nos suministra el lenguaje C.

2.2.1. ¿Qué son los punteros?: Desmitificando al monstruo

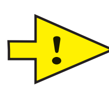
Un puntero no es más que una variable que cuyo contenido, en vez de datos, son direcciones de memoria.

Dicho de otra forma: Hasta ahora utilizábamos las variables para guardar cantidades numéricas, caracteres o cadenas de texto. Ahora utilizaremos un tipo especial de variable que será capaz de guardar una dirección de memoria.

Contemplemos el siguiente esquema que nos resultará familiar del anterior artículo (PC PASO A PASO 12):



En la primera línea resaltada char *var_puntero;, lo que hacemos es declarar una variable de tipo puntero que es capaz de apuntar a direcciones de memoria (es decir, es capaz de guardar direcciones de memoria), que contengan un dato char. Esto puede causar confusión dado que si sabemos que los punteros contienen direcciones de memoria, ¿qué necesidad hay de asociarlos un tipo? Esta pregunta, que en principio puede parecer compleja, queda explicada cuando más adelante veamos que como el resto de las variables podemos realizar operaciones aritméticas con el contenido de las mismas, como por ejemplo el incremento. Claro que no es lo mismo incrementar una dirección que apunta a caracteres la cual se incrementará (siempre según la figura) de un byte en un byte, que incrementar la dirección que apunta a un número real, que aumentará de cuatro en cuatro bytes. Es decir, el puntero "debe conocer a qué apunta".



Aunque esta...

Aunque esta explicación no sea muy formal, creo que sirva para comprender alguno de los comportamientos particulares de los punteros.

No es la primera vez que en esta serie de artículos se prescinde del rigor informático en favor de la claridad de la exposición del tema. Por poner un ejemplo, en el esquema de arriba (ya presente en el anterior artículo) existe un gazapo muy sutil. Está relacionado en la manera en como se guardan los números reales en la memoria de un ordenador. Dejo al lector la tarea de encontrar este gazapo. En esta búsqueda, espero, aprenderá y comprenderá mucho sobre las representaciones de datos en la memoria de una computadora.

Tras examinar esta línea llegamos a la conclusión de que un puntero se declara siempre según el siguiente esquema:

```
tipo_dato *nombre_variable_puntero;
```

Como vemos es el asterisco precediendo al nombre de la variable el que convierte a esta en un puntero: El asterisco, en el caso de la declaración de variables, es quien dice si la variable es un puntero y no una variable normal.

Otros ejemplos de declaración de punteros serán:

```
int   *p_entero;
float *p_real;
char  *p_caracter;
```



Existe un tipo de...

Existe un tipo de puntero denominado void que virtualmente en conjunción con el casting es capaz de apuntar a cualquier tipo de variable o dirección de memoria. Recomiendo que el lector investigue por su cuenta este tipo de punteros pues muy pronto los veremos. Ya sabes: www.google.com ;)

2.2.2. Operadores sobre punteros

Dos son los principales operadores que se utilizan en relación con los punteros:

& (ampersand)

El operador & nos devuelve la dirección de memoria donde se encuentra una determinada variable.

Algunos ejemplos de este operador son:

```
var_puntero=&caracter;

/* El ejemplo visto arriba. A var puntero le asignamos la dirección
de memoria donde reside caracter. En la figura la variable
caracter se encuentra en la posición 00, luego var_puntero
contendrá dicha dirección (apuntará a dicha dirección) */
int cosa1, cosa2;
int *pent;
pent=&cosa1;
printf("La dirección de cosa1 es %m", &cosa1);
pent=&cosa2;
printf("La dirección de cosa2 es %m y el puntero apunta a %m",&cosa2,pent);
printf("El puntero pent que está en %m apunta a %m", &pent, pent);
```

*** (asterisco)**

Este operador nada tiene que ver con lo explicado arriba sobre como se declara una variable puntero.

Este operador que precede siempre a una variable puntero, nos devuelve el contenido de la dirección almacenada por el puntero. Ojo: El contenido de la dirección almacenada por el puntero. Por poner un ejemplo, imaginemos que el código de la figura lo cambiamos a:

```
#include <stdio.h>
main()
{
   char caracter;
   char *var_puntero;
   /* código ignorado por carecer de importancia */
   caracter=10;
   var_puntero=&caracter;
   printf("%c", *var_puntero);
}
```

En este programa imprimiremos lo que hay almacenado en la dirección que contiene la variable puntero; es decir, en la dirección donde apunta el puntero. Como hemos hecho que el puntero contenga la dirección (apunte) de la variable caracter (la dirección 00 en el diagrama) en la instrucción var_puntero=&caracter, el contenido de lo apuntado por el puntero será el contenido de la variable a la que apunta el puntero: el caracter correspondiente al decimal 10 (salto de línea).

A este acceso al contenido de una variable mediante un puntero se le denomina indirección.

A continuación muestro un pequeño programa que puede llegar a facilitar la comprensión de los operadores & y *

```
/*
 * Programa: ej01.c
 *
 * Descripción:
 *    Este programa muestra  el comportamiento básico de los punteros.
 *
 * Compilación:
 *    gcc ej01.c -o ej01
 */
#include <stdio.h>
main()
{
      char vchar1, vchar2;
      char *pchar1, *pchar2;
      /* Inicializamos las variables */
      vchar1='a';
      vchar2='b';
      /* Inicializamos los punteros */
      pchar1=&vchar1;
      pchar2=&vchar2;
      /* Imprimimos información */
      printf("\n Tras ejecutar las siguientes instrucciones:\n\n");
      printf("\t\033[31;1m vchar1='a';\n");
      printf("\t vchar2='b';\n\n");
      printf("\t pchar1=&vchar1;\n");
      printf("\t pchar2=&vchar2;\033[0;0m\n\n");
      printf(" El contenido de las variables y los punteros\n");
      printf(" es el siguiente:\n");
      printf("\n \033[34;1m Dir Variable \t\t Contenido\t\tContenido de lo apuntado \033[0;0m");
      printf("\n &vchar1 = %x,\t vchar1 = %c", &vchar1, vchar1);
      printf("\n &vchar2 = %x,\t vchar2 = %c", &vchar2, vchar2);
      printf("\n &pchar1 = %x,\t pchar1 = %x,\t *pchar1 = %c", &pchar1, pchar1, *pchar1);
      printf("\n &pchar2 = %x,\t pchar2 = %x,\t *pchar2 = %c", &pchar2, pchar2, *pchar2);
      printf("\n");
}
```

Tras su ejecución, obtendremos una salida similar a esta:

```
luis@nostromo:~/hxc/articulo6_el_chaman$ ./ej01

Tras ejecutar las siguientes instrucciones:
      vchar1='a';
      vchar2='b';
      pchar1=&vchar1;
      pchar2=&vchar2;

El contenido de las variables y los punteros es el siguiente:

 Dir Variable             Contenido           Contenido de lo apuntado

&vchar1 = bffff9f7,     vchar1 = a

&vchar2 = bffff9f6,     vchar2 = b

&pchar1 = bffff9f0,     pchar1 = bffff9f7,     *pchar1 = a

&pchar2 = bffff9ec,     pchar2 = bffff9f6,     *pchar2 = b
```



Ejercicio 0.1

Ejercicio 0.1

Teniendo el siguiente código:

```
/*
 * Programa: ej02.c
 *
 * Descripción: Segundo ejercicio del artículo 6
 *
 * Compilación:
 *    gcc ej02.c -o ej02
 *
 */
#include <stdio.h>

main()
{
  int a, b, *pInt;

  ........
  ........
  ........
  ........

  printf("\n  a = %i, b = %i \n", a, b);
}
```

Sustituir las líneas punteadas por las instrucciones necesarias para que el resultado sea:

luis@nostromo:~/hxc/articulo6_el_chaman$ ./ej02

a = 3, b = 5

¡Ah! Se me olvidaba. Para este ejercicio están prohibidas las siguientes instrucciones:

a = 3;
b = 5;



Solución del ejercicio

Solución del ejercicio

```
/*
 * Programa: ej02.c
 *
 * Descripción: Segundo ejercicio del artículo 6
 *
 * Compilación:
 *    gcc ej02.c -o ej02
 *
 */
#include <stdio.h>
main()
{
        int a,b, *pInt;

        pInt=&a;
        *pInt=3;
        pInt=&b;
        *pInt=5;
        printf("\n a = %i, b = %i \n",a,b);
}
```

2.2.3. Operando con punteros

Anteriormente hemos mencionado que los punteros se pueden incrementar y decrementar. Cuando hagamos esto tendremos que tener presente que lo que se incrementa o decrementa es el contenido del puntero, o dicho de otra forma: una dirección de memoria.

Prestemos atención a este código:

```
/*
 * Programa: ej03a.c
 *
 * Descripción:
 *   Muestra el uso de los operadores incremento y decremento
 *   con un puntero.
 *
 * Compilación:
 *   gcc ej03a.c -o ej03a
 *
 */
include <stdio.h>
main()
{
    /* Declaramos las variables necesarias */
    int vector[5];
```

```
    int i, *pEntero;
    /* Llenamos el vector con los números 1, 2, 3, 4 y 5 */
    for(i=0;i<5;i++)
    {
        vector[i]=i+1;
    }
    /* Hacemos que el puntero apunte al array */
    pEntero = vector;
    printf(" %i \n", *pEntero);
    pEntero++; /* Equivale a pEntero = pEntero +1 */
    printf(" %i \n", *pEntero);
    pEntero++;
    printf(" %i \n", *pEntero);
    pEntero++;
    printf(" %i \n", *pEntero);
    pEntero++;
    printf(" %i \n", *pEntero);
}
```

En este código lo primero que nos debería de llamar la atención es la instrucción pEntero = vector;. Es fácil pensar que la instrucción adecuada sería pEntero = &vector;. Pero no olvidemos que pEntero es un puntero que apunta a una variable de tipo entero mientras que vector representa un conjunto de variables de tipo entero que sólo puede accederse a ellas a través de los índices.

Para rizar más el rizo, el programa sería igual de válido si en vez de escribir pEntero = vector;, hubiésemos escrito pEntero = &vector[0];.

Y esto es porque los vectores y los arrays en general, a efectos prácticos, son punteros... Y viceversa...

Al ir aumentando de uno en uno el puntero, podría parecer que estamos incrementando en un byte la dirección de memoria, pero el siguiente código nos va a sacar de dudas:

```
/*
 * Programa: ej03b.c
 *
 * Descripción:
 *   Muestra el uso de los operadores incremento y decremento
 *   con un puntero, así como las peculiaridades de cada tipo
 *   de puntero.
 *
 * Compilación:
 *   gcc ej03b.c -o ej03b
 *
 */
#include <stdio.h>
main()
{
    /* Declaramos las variables necesarias */
    int   vectorE[5];
    int   i, *pEntero;
    char  vectorC[5], *pCaracter;
    float vectorR[5], *pReal;
    /* Llenamos los  vectores con datos */
    for(i=0;i<5;i++)
    {
        vectorE[i]=i+1;
        vectorC[i]=i+40;
        vectorR[i]=(float)i*0.1;
    }
    /* Hacemos que el puntero apunte al array */
    pEntero   = vectorE;
    pCaracter = vectorC;
    pReal     = vectorR;
    printf("\n Los puntero se incrementarán es:");
    printf("\n  pEntero:   %i bytes", sizeof(int));
    printf("\n  pCaracter: %i bytes", sizeof(char));
    printf("\n  pReal:     %i bytes", sizeof(float));
    printf("\n Cada vez que pongamos \"puntero++\"\n");
    printf(" %i, %c, %f \n", *pEntero, *pCaracter, *pReal);
    pEntero++;    /* Equivale a pEntero   = pEntero +1   */
    pCaracter++;  /* Equivale a pCaracter = pCaracter +1 */
    pReal++;      /* Equivale a pReal     = pReal +1     */
    printf(" %i, %c, %f \n", *pEntero, *pCaracter, *pReal);
    pEntero++;
    pCaracter++;
    pReal++;
    printf(" %i, %c, %f \n", *pEntero, *pCaracter, *pReal);
    pEntero++;
    pCaracter++;
    pReal++;
    printf(" %i, %c, %f \n", *pEntero, *pCaracter, *pReal);
    pEntero++;
    pCaracter++;
    pReal++;
    printf(" %i, %c, %f \n", *pEntero, *pCaracter, *pReal);
}
```

¿Por qué sabe cada puntero sabe en cuántas casillas debe incrementarse cuando nosotros tan sólo le incrementamos en una posición? Pues porque nosotros hemos dicho a cada puntero a qué familia de punteros pertenece al declararlos (int, float, char...., de manera que automáticamente los punteros se incrementarán tantos bytes como ocupen en memoria sus tipos base.

Otra particularidad de poder incrementar el contenido de un puntero, es la utilización de éste como si de un vector se tratase. Veámoslo, porque es cuando menos curioso:

```
/*
 * Programa: ej03c.c
 *
 * Descripción:
 *    Muestra el comportamiento de un puntero como vector
 *
 *
 * Compilación:
 *    gcc ej03c.c -o ej03c
 *
 */
#include <stdio.h>
main()
{
      /* Declaramos las variables necesarias */
      int vector[5];
      int i, *pEntero;
      /* Llenamos el vector con los números 1, 2, 3, 4 y 5 */
      for(i=0;i<5;i++)
      {
            vector[i]=i+1;
      }
      /* Hacemos que el puntero apunte al array */
      pEntero = vector;
      /* Comportamiento de un puntero como un array "normal" */
      for(i=0;i<5;i++)
      {
            printf(" %i ", pEntero[i]);
      }
      printf("\n");
      /* Comportamiento de un puntero como "base + indirección" */
      for(i=0;i<5;i++)
      {
            printf(" %i ", *(pEntero + i));
      }
}
```

Obsérvese que en el segundo bucle for, pEntero se comporta como si de un array se tratase, accediendo al contenido de las distintas posiciones de memoria mediante pEntero[i] donde i es el índice. Observando el tercer bucle for podemos llegar a la conclusión de que pEntero[i] es una manera abreviada de escribir *(pEntero + i).... Y no estaremos en absoluto equivocados.

3. Cierre

Y con esto termina el artículo de hoy. Ya sólo nos quedan un par de cosillas del lenguaje C (estructuras, uniones, campos de bits, funciones, etc....) y pronto podremos meternos en el C puro y duro.

Puede parecer que en este artículo hemos dejado en la cuneta la parte de la programación modular. Considero que no ha sido así pues en este artículo ya se manda realizar alguna práctica con lo que espero que os vayáis familiarizando con el compilador de GNU.

En el número que viene, explicaremos el tema de las funciones con un pequeño proyecto que ya usará Makefiles, archivos de cabecera y, como no, funciones, estructuras de datos, punteros, muchos punteros y alguna que otra macro.

Obviamente, antes de enfrentarnos a todo esto, debemos de tener muy claro que lo que vamos viendo lo debemos de manejar con cierta soltura. Espero que este artículo os ayude a ello. Nos vemos en el foro. Saludos y gracias por vuestra atención.

Bibliografía

man printf; man scanf, Varios.

UNiX Programación Avanzada, Fco. Manuel Márquez, Editado por Ra-Ma, ISBN: 84-7897-239-0.

Slackware Linux Unleashed, Bao Ha y Tina Nguyen, Editado por Pearson Education, ISBN: 0-672-31768-0.

Advanced Linux Programming, Mark Mitchel, Jeffrey Oldham, y Alex Samuel, Editado por New Riders, ISBN: 0-7357-1043-0.

CURSO DE VISUAL BASIC
UN CLIENTE, UNA NECESIDAD, TENEMOS UN PROYECTO (PARTE I)

VISUAL BASIC es la forma más rápida de crear un programa. Vamos a aplicar los conocimientos acumulados y a crear un " proyecto real".

Aquí estamos de nuevo con nuestro particular curso de Visual Basic. Después de varios meses escribiendo para vosotros he decido que la temática ha llegado a su fin, con esto quiero decir que esta será la última entrega del curso de aprendizaje de Visual Basic y constará de dos o tres partes.

Os quiero recordar que la finalidad de esta serie de artículos era despertaros el gusanillo de la programación y animaros a crear vuestros propios programas. Ciertamente, no hemos dedicado mucho tiempo a ningún tema relacionado con la seguridad, y creo que ha sido lo correcto, ya que la revista dedica gran parte de su contenido a esta temática, y si dejamos a parte la programación como tal, estaríamos montando una mesa con tres patas.

Por eso, en este curso, hemos aprendido Visual Basic desde cero 0, y hemos ido subiendo el nivel progresivamente. Estoy seguro que vosotros habéis dedicado horas de vuestro tiempo libre para realizar los ejercicios y ampliarlos a vuestro gusto, eso me enorgullece. Así que, para acabar y despedirnos, plantearemos un ejercicio con acceso a base de datos, diseño de formularios, organización, etc, etc...

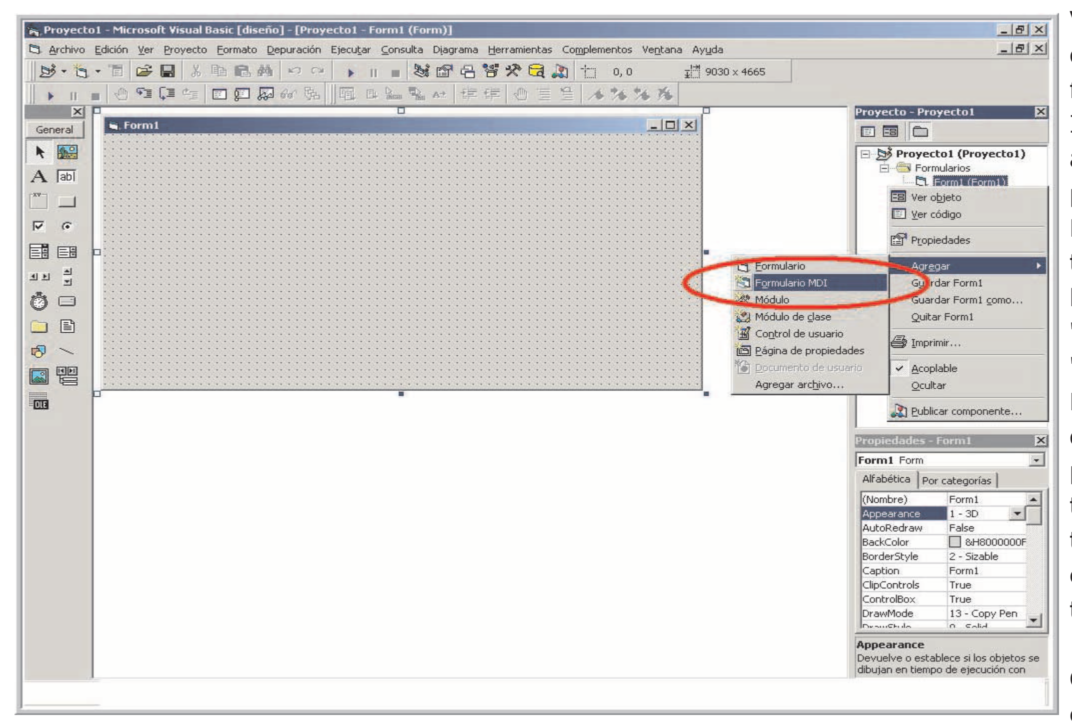
Y es que nos vamos a imaginar a un cliente que tiene una necesidad, y nosotros, como proveedores vamos a darle una solución a un altísimo coste, claro... ;)

Nuestro cliente podría ser Micosoft (conocida marca de helados), y necesitan un software, medianamente decente, que les gestione el stock de helados. Nosotros, como proveedores, damos una respuesta comercial, y le indicamos que sobradamente podemos abarcar el proyecto. Ellos nos comentan que necesitan un software de gestión que catalogue los helados por su nombre, y si fuera necesario, su sabor. También cree necesario que se controlen las entradas de productos y, como no, las ventas de estos. La base de datos debe estar en un servidor central, mientras que las terminales deben poder interactuar con él. El programa se instalará en diferentes puntos de venta, así como en el almacén, desde donde se controlarán las entradas. Por último, nos piden que elaboremos un módulo capaz de crear estadísticas sobre las ventas.

Parece difícil, ¿no?, pues veréis que realmente, así es, difícil como lo parece, sobre todo porque el cliente nunca queda satisfecho.

Empecemos pues. Lo primero que voy a hacer es introduciros en un nuevo concepto de organización de formularios, el menú conocido como MDI.

El MDI no es más que un "padre de formularios", es decir, un formulario gigante que abarca todos los demás. Cuando creamos un formulario MDI, es lógico hacerlo padre de los demás formularios que creemos posteriormente. Abramos un nuevo proyecto. Lo primero que vamos a hacer es agregar el formulario de tipo MDI en el explorador de proyectos. Lo haremos picando con el botón derecho sobre él y eligiendo la opción "Formulario MDI" del menú "agregar".



Vamos ahora al cuadro de propiedades del formulario MDI. Indiquémosle que se abra, por defecto, a pantalla completa. Para hacer esto tenemos que cambiar la propiedad "WindowsState" a "Maximized". Ejecutamos y vemos que aparece una pantalla que ocupa todo nuestro área de trabajo, exceptuando claro está, la barra de tareas.

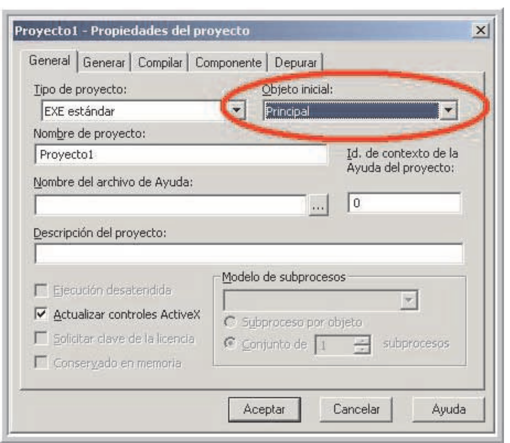
Apreciaremos que se nos ha agregado al proyecto un formulario más oscuro de lo normal. Tenemos que decirle al proyecto que este será nuestro formulario principal, y que los demás son hijos de él.



Posteriormente...

Posteriormente veremos como hacer que los formularios sean hijos (child) de un MDI.

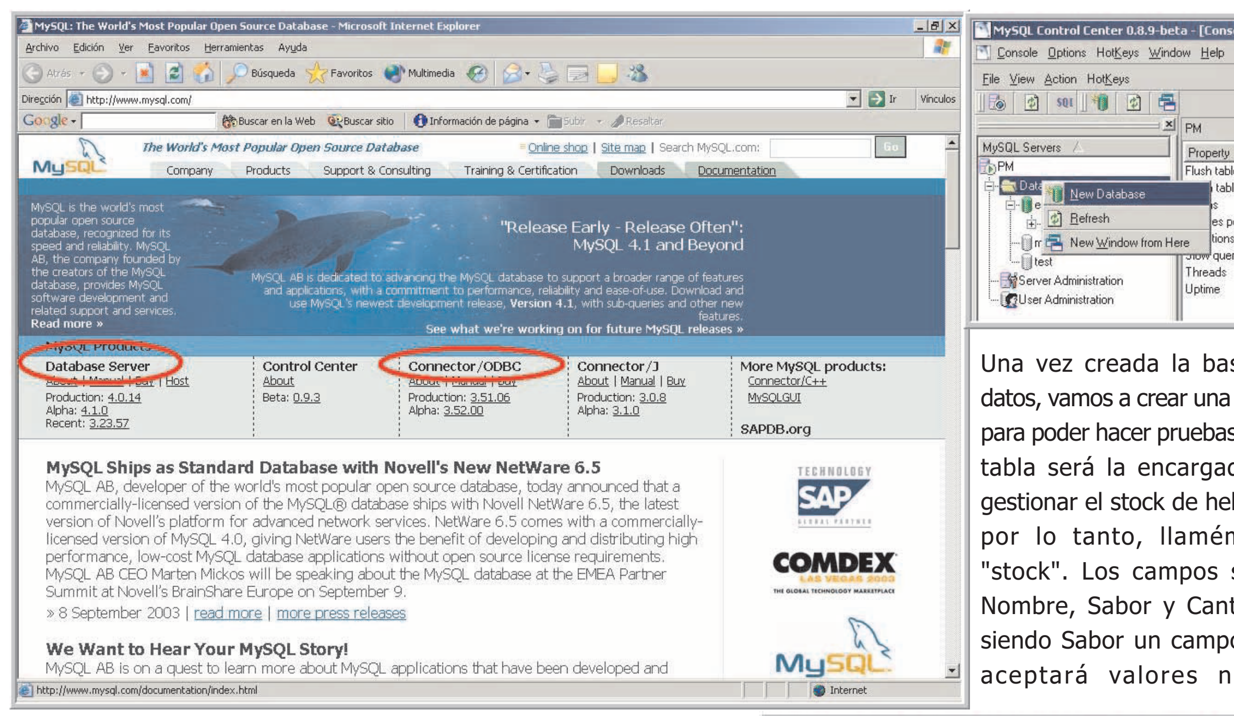
Para indicar al proyecto que debe iniciarse por uno u otro formulario, vamos al menú "Proyecto", "Propiedades" y en el combo "Objeto inicial" seleccionamos el nombre de nuestro formulario MDI (al cual yo he llamado



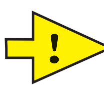
Cabe decir que estamos empezando la casa por el tejado, pero quería dejar claro el concepto de formulario MDI antes de empezar a diseñar nuestra base de datos.

Para vuestra sorpresa, dejaremos el Access y nos introduciremos en el maravilloso mundo de MySql, que aunque parezca increíble, también se pueden gestionar desde Visual Basic. Voy a intentar hacer un resumen de cómo instalarse una base de datos MySql en local. Seré breve, ya que existen cientos de manuales que lo explican detalladamente, y porque si lo explicase todo con pelos y señales, nos ocuparía prácticamente toda esta entrega.

Bien, existen varias formas de tener y gestionar MySql en tu PC. Una es instalando directamente el gestor que podemos encontrar en http://www.mysql.org (el "database server") y usar el "Control Center" que podemos encontrar en la misma página. Otra opción es descargar el servidor web Apache, instalarlo y descargar también el conocido "PHPMyAdmin", conjunto al script necesario para hacer funcionar PHP en Apache web Server. Por supuesto esta opción es mucho más complicada, pero lo dejo a vuestra elección. Lo que si debemos descargar obligatoriamente en los dos casos es el "Database Server" y el "Conector/ODBC", que nos va a ayudar a crear la conexión con la base de datos.



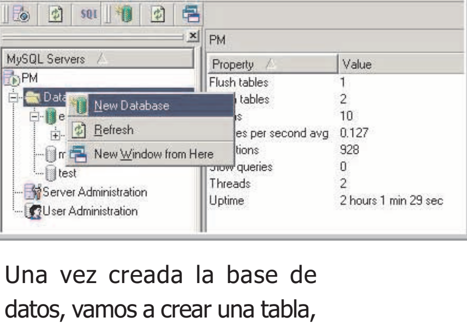
Una vez descargados, debemos instalar el servidor de base de datos, el driver de ODBC y el "Control Center", en este orden a ser posible. Una vez instalado todo, y si lo hemos hecho correctamente, deberíamos tener en la barra de tareas el icono de un semáforo, con la luz verde encendida, que viene a indicar que está arrancado el servicio de MySql.



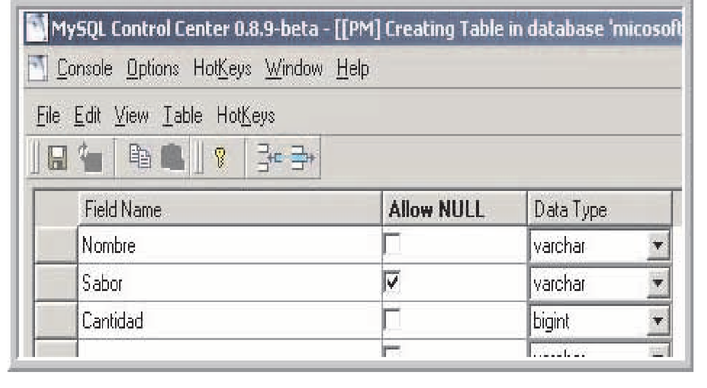
En el curso de apache...

En el curso de APACHE, publicado en los anteriores números de la revista, ya instalamos el MySQL paso a paso. De todas maneras, si tienes problemas durante la instalación, recuerda que en el foro de la revista (www.hackxcrack.com) puedes contactar con muchos de nuestros lectores y hacer las consultas que quieras, seguro que recibes la ayuda que necesitas :)

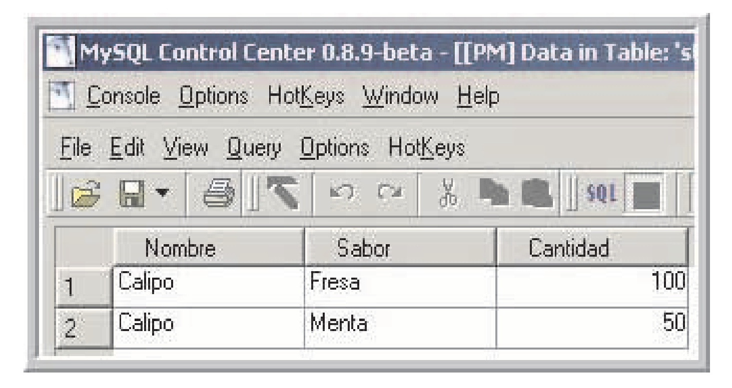
Ejecutamos entonces el "Control Center" y deberíamos ver un listado de todas las bases de datos de prueba que MySql nos ha creado. Vamos a crear nuestra base de datos. Con el botón derecho en el explorador de bases de datos, seleccionamos "New Database" y la llamamos micosoft.



Una vez creada la base de datos, vamos a crear una tabla, para poder hacer pruebas. Esta tabla será la encargada de gestionar el stock de helados, por lo tanto, llamémosla "stock". Los campos serán Nombre, Sabor y Cantidad, siendo Sabor un campo que aceptará valores nulos.



Introduzcamos algunos registros para poder hacer una prueba primero de acceso a la base de datos, por ejemplo, con un par de ellos, tenemos de sobra.

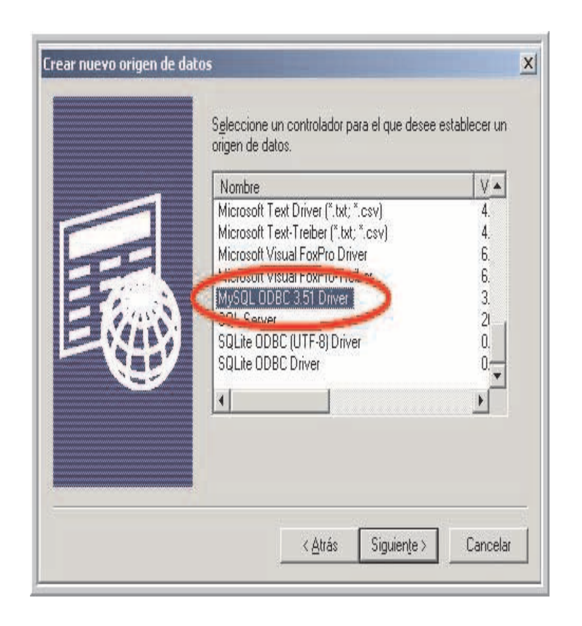


	Nombre	Sabor	Cantidad
1	Calipo	Fresa	100
2	Calipo	Menta	50

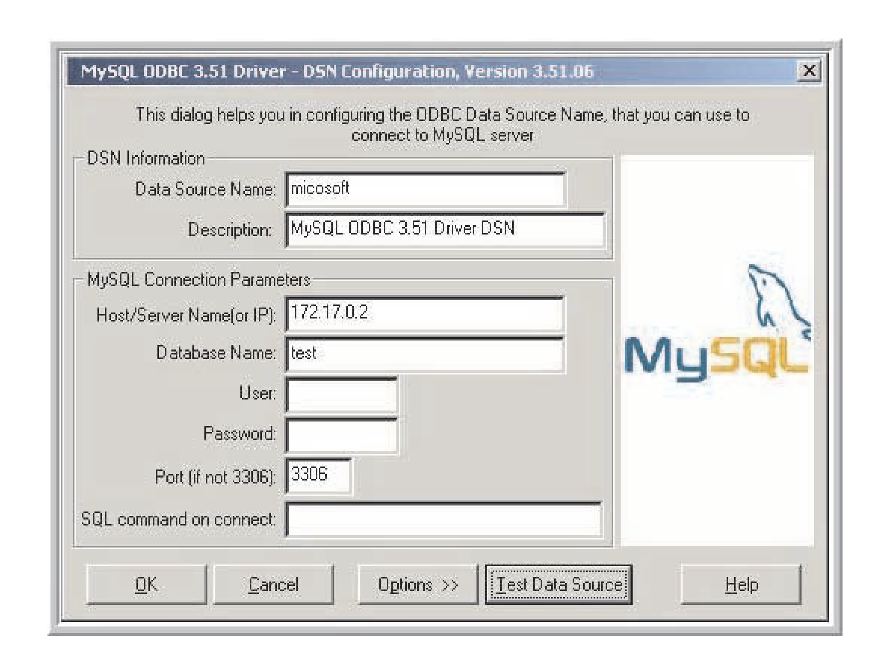
Muy bien. Ahora intentemos un acceso fugaz a la tabla. Volvamos al proyecto de Visual Basic y agreguemos las referencias de acceso a

datos (Microsoft Active Data Objects) y el ADO DataControl. También vamos a agregar el "Microsoft Windows Common Controls 6.0".
Vamos a ayudarnos con el ADO Data Control para generar una cadena de conexión que utilizaremos temporalmente para esta prueba. Creo que ya expliqué los pasos a seguir para generar una cadena de conexión, pero los volveré a explicar. Añadimos el Data Control. Vamos a la propiedad "ConnectionString" y picamos en el botón con puntos que aparece. Nos debería aparecer un pequeño formulario con una opción para generar la cadena de conexión, seleccionamos esa.

Nos volverá a aparecer otro formulario con diferentes proveedores OLE DB, ignorémoslos. Vamos a la segunda pestaña y volvemos picar en "Usar cadena de conexión", "Generar". En la siguiente ventana elegimos la segunda pestaña y picamos en "Nuevo". Elegimos la segunda opción, "Orígenes de datos del sistema", para que todo el que se conecte al equipo lo pueda utilizar. En la siguiente ventana, elegimos el driver correspondiente a MySql, que seguramente estará de los últimos, y le ponemos un nombre al origen de datos.



También podríamos introducir la dirección IP en vez del literal "LocalHost" en el cuadro de configuración, y testear la conexión inmediatamente después (no os olvidéis de poner el nombre de vuestra BD, no dejéis puesta la que viene por defecto, "test").





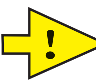
La dirección IP...

La dirección IP que se aparece en la imagen es una dirección de red interna. Para no tener problemas de acceso, yo introduciré el literal "LocalHost".

Aceptamos varias veces, hasta llegar al punto en que tenemos una preciosa cadena de conexión a la base de datos. Agregamos un módulo a nuestro proyecto, un módulo que contendrá, por ahora, una variable global con la cadena de conexión, una variable de tipo Connection y otra Recordset, además de una función para inicializarlos.

```
Option Explicit
Global Cadena As String
Global Conn As ADODB.Connection
Global Rs As ADODB.Recordset

Public Function Conectar() As Long
On Error GoTo Errores
   Cadena = "DSN=micosoft;DESC=MySQL ODBC 3.51 Driver
   DSN;DATABASE=micosoft;SERVER=localhost;UID=;PASSWORD=;PORT=3306;
   OPTION=3;STMT=;"
   Set Conn = New ADODB.Connection
   Conn.ConnectionString = Cadena
   Conn.Open
   Exit Function
Errores:
   Conextar = Err.Number
End Function
```



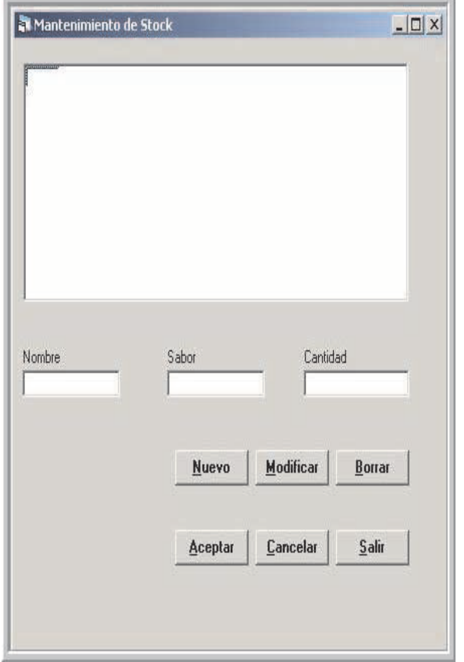
La función...

La función "On Error goto Errores" es la manera que tiene el Visual Basic de controlar los errores. Esta función está indicando que, en caso de que se produzca un error en la conexión a la base de datos, el código haga un salto de línea hasta la etiqueta "Errores", donde podemos ver que inmediatamente después hace que el valor de la función sea el número de error que se ha producido, ya que el objeto "Err" nos puede indicar tanto el número como la descripción. Un ejemplo para entender mejor esto sería provocar un error voluntariamente (cambiando caracteres de la cadena de conexión, por ejemplo) y poniendo un "MsgBox Err.Descripción", recibiendo así un mensaje con la descripción del error.

Para ver que no hay errores, llamaremos a la función abrir desde el "Form_Load" del MDI.

```
Option Explicit
Dim Error As Integer

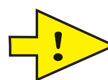
Private Sub MDIForm_Load()
    Error = Conectar
End Sub
```



¿Va bien?, perfecto. Vamos a diseñar nuestro primer formulario, que será en parte nuestro formulario de prueba. Este nos mostrará el Stock de productos, además de permitir dar de alta, modificar y borrar estos. Al formulario agregaremos un objeto "ListView" que nos muestre las cantidades de helados que tenemos, separadas por marcas.

También agregaremos algunos Labels y botones, para poder realizar mantenimientos del stock. Este es uno de los posibles aspectos del formulario.

Volvemos un segundo al MDI. La función principal, como ya he dicho, del MDI es actuar como menú principal del proyecto. Ya que esta es su función, debemos agregar los módulos necesarios para poder abrir los formularios según el usuario final lo necesite. Para hacer este acceso más atractivo, nosotros vamos a agregar una "ToolBar". La podemos añadir simplemente haciendo doble click sobre ella en el cuadro de herramientas, ya que se posicionará automáticamente en su lugar. Vamos ahora a su cuadro de propiedades y picamos sobre "personalizado". En la primera pestaña tenemos una gran cantidad de opciones que, principalmente, cambian el diseño de la "ToolBar". A nosotros nos interesa la segunda pestaña, "Botones". Vamos agregando botones a la barra, usando el botón "Insertar Botón", pudiendo a su vez incluir iconos para dar un aspecto más atractivo a la "ToolBar".



Para agregar...

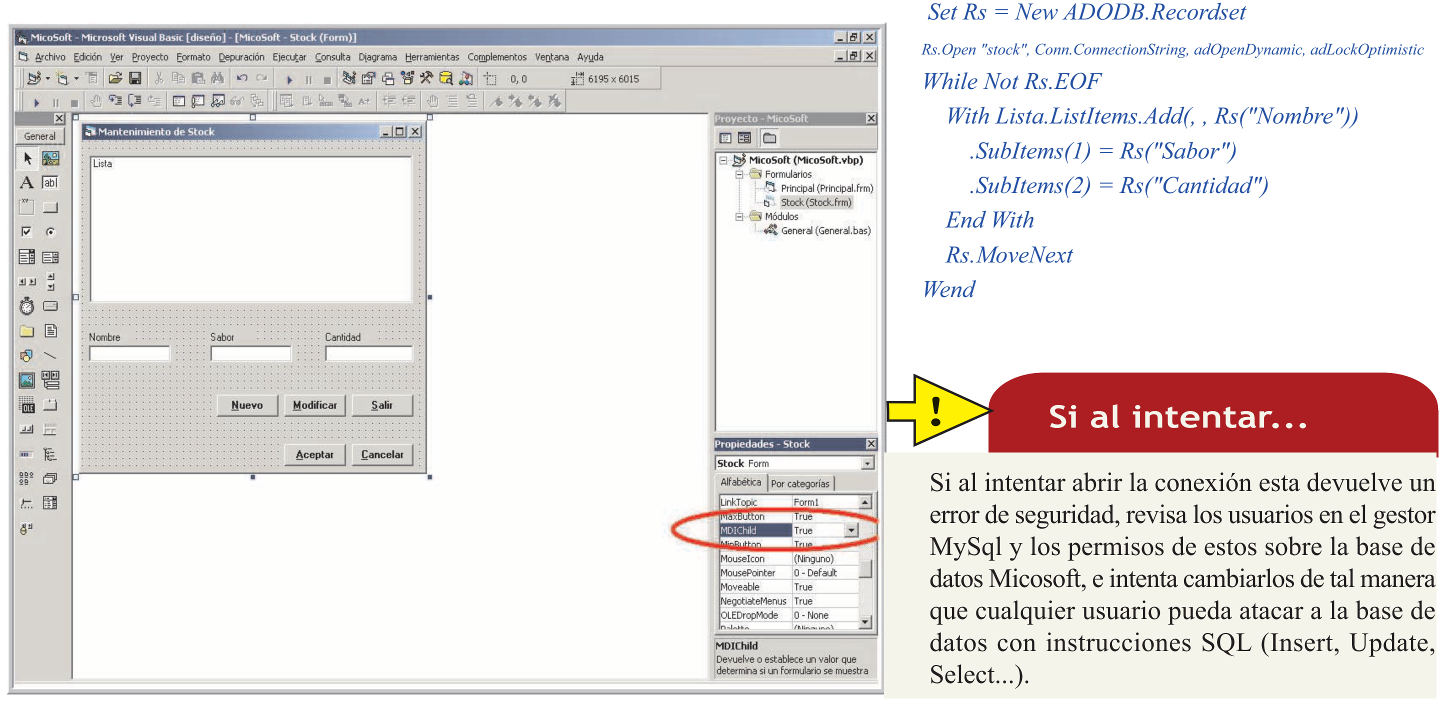
Para agregar iconos o imágenes a la ToolBar es necesario antes añadir al formulario un objeto del tipo "ImageList" y agregar aquí las imágenes. Posteriormente podemos indicar a la ToolBar que va a utilizar esta ImageList en la primera pestaña, e indicar también que imagen corresponde a que botón en la pestaña "Botones", propiedad "Image"

Una vez acabada la barra de herramientas con los menús, vamos a indicarle que, al pulsar stock, se abra el formulario que estamos diseñando. Vamos entonces a codificar el evento "Click" de la ToolBar. La cosa quedaría así.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
    Select Case Button.Caption
        Case "Stock"
            Stock.Show
    End Select
End Sub
```

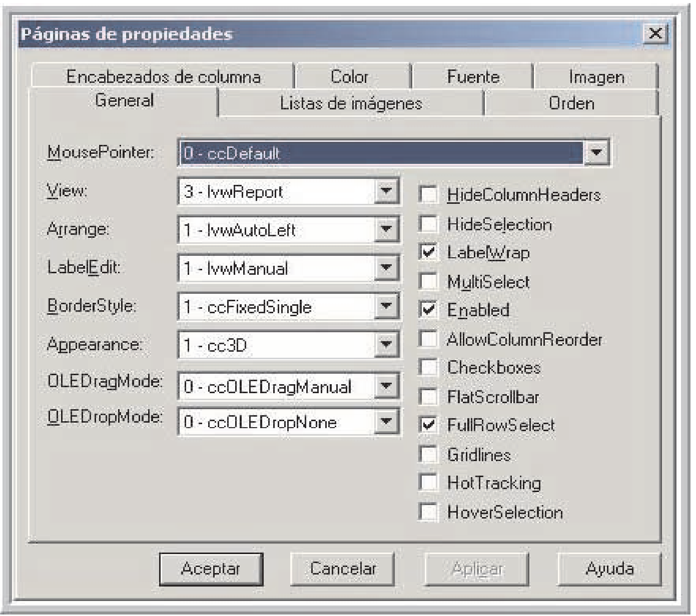
Yo he decidido utilizar un Select Case, pero sería totalmente válido hacerlo con sentencias condicionales de tipo "If". Ejecutamos y vemos que al pulsar sobre "Stock" se nos abre el formulario, pero lo hace de manera independiente, como ignorando al MDI, ya que si cerramos el MDI, el formulario de Stock seguiría abierto, y esto no es lo que hemos dicho. Lo que aquí falta es que le digamos a

la venta de stock que es hija del principal. Esto se hace poniendo a verdadero la propiedad "MDIChild" que todo formulario tiene, y en nuestro caso, lo aplicaremos al formulario "stock" y a todos los que vayamos agregando a partir de ahora.



Sigamos en este formulario. Vamos a las propiedades de la lista y cambiamos la propiedad "View" a "lvwReport", para que el diseño del ListView visualice al estilo"detallado".

También podemos cambiar más propiedades de la lista, por ejemplo, en esta imagen vemos como la tengo configurada.



El siguiente paso sería mostrar todos los datos contenidos en la tabla "Stock" por la lista. El código necesario para esto sería el siguiente, que pasamos a comentar

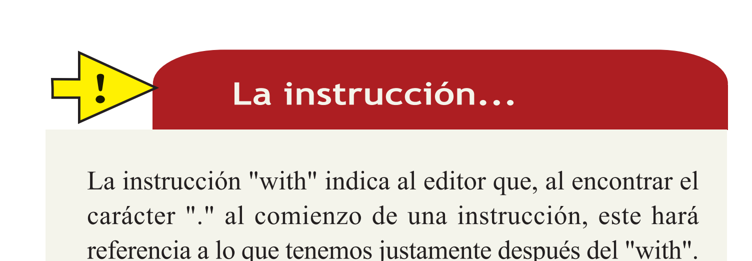
```
Set Rs = New ADODB.Recordset
Rs.Open "stock", Conn.ConnectionString, adOpenDynamic, adLockOptimistic
While Not Rs.EOF
    With Lista.ListItems.Add(, , Rs("Nombre"))
        .SubItems(1) = Rs("Sabor")
        .SubItems(2) = Rs("Cantidad")
    End With
    Rs.MoveNext
Wend
```

Si al intentar...

Si al intentar abrir la conexión esta devuelve un error de seguridad, revisa los usuarios en el gestor MySql y los permisos de estos sobre la base de datos Micosoft, e intenta cambiarlos de tal manera que cualquier usuario pueda atacar a la base de datos con instrucciones SQL (Insert, Update, Select...).

Es bastante sencillo de entender. La primera línea instancia el objeto Recordset, mientras que la segunda lo abre. Inmediatamente después creamos un bucle que va añadiendo "Items" a la lista hasta que lleguemos al final del fichero. Vemos que los datos a mostrar son Nombre, Sabor y Cantidad.

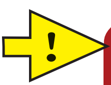
No se nos debe olvidar mover el objeto Recordset al siguiente registro (Rs.MoveNext), de lo contrario estaríamos creando un bucle infinito que acabaría provocando un desbordamiento de memoria



La instrucción...

La instrucción "with" indica al editor que, al encontrar el carácter "." al comienzo de una instrucción, este hará referencia a lo que tenemos justamente después del "with". Por ejemplo, si pusiéramos la línea "With Rs" cada vez que pulsáramos el carácter "." Se nos desplegaría la lista de propiedades del Recordset "Rs". La instrucción "with" se cierra con "End with"

Mmmmm..., ahora que hemos llegado a este punto, me doy cuenta de que falta algo en la base de datos. Para ser más exactos, un campo clave en la tabla stock. Y es que una tabla sin campos clave es como un coche sin luces, funciona pero es poco recomendable.



¿Qué es un...

¿Qué es un "campo clave"? Para quienes trabajan normalmente con bases de datos, esta explicación sobra; pero hay muchas personas que nunca han creado una base de datos y seguro que agradecerán esta nota. Sin entrar en tecnicismos diremos que un **CAMPO CLAVE es simplemente un campo existente de la tabla creada que no puede en ningún caso contener valores repetidos.**

Para que se entienda: Si estuviésemos creando una agenda de nuestros amigos con los campos "nombre" (en este campo introduciríamos los nombres de nuestras amistades), campos "apellido" (en este campo introduciríamos los apellidos correspondientes a cada nombre) y campos "DNI" (en este campo introduciríamos el DNI correspondiente a cada nombre), **el "campo clave" debería ser el campo "DNI"**. Está claro que no existirán dos personas con el mismo DNI :)

Un par de apuntes:
1.- Imagina que en nuestra agenda no queremos que figuren los DNI. Nos quedarían dos campos ("nombre" y "apellido") y no podemos hacer que uno de ellos sea campo clave porque podemos tener a dos amigos que se llamen Pedro o dos amigos que se apelliden González… ¿Qué hacemos entonces? Pues la practica habitual es crear un nuevo campo (llamado por ejemplo "identificador") donde pondremos una serie auto-numérica.

¿Qué es una serie auto-numérica? Pues muy sencillo, cuando introduzcamos PEDRO en el campo nombre, automáticamente rellenará el campo "identificador" con un 1, cuando introduzcamos el nombre de un nuevo amigo, automáticamente se rellenará al campo "identificador" con un dos, el siguiente con un tres y así hasta que nos cansemos. De esta forma se consigue un CAMPO CLAVE y nos despreocupamos de todo puesto que encima es automático.

	Campo identificador	Campo "nombre"	Campo "apellido"
Registro 1	1	Alberto	García Gómez
Registro 2	2	Maria	Fernández Dalma
Registro 3	3	Alberto	Matamoros Dalma
.			
.			
.			
Registro 500	500	Perico	Palote Pinto
.			
.			

2.- También puedes establecer lo que se llama un campo clave combinado. Es cuando la combinación de dos campos (por ejemplo la combinación del campo "nombre" y campo "apellido") nunca puede dar un valor repetido. En este caso no se cumple, porque podemos tener dos amigos que se llamen igual y tengan los mismos apellidos.

Por eso vamos a analizar cual podría ser nuestro campo clave. Sólo disponemos de 3. El primero, Nombre, por si solo no puede serlo, ya que pueden haber varios helados con la misma marca (o nombre) y diferentes sabores, así que rompería la lógica de un campo clave. El segundo tampoco, ya que puede haber decenas de helados con sabor a fresa, y el campo cantidad es absurdo. ¿Qué hacer entonces?, pues tenemos varias posibilidades, pero yo voy a exponer aquí las que encuentro más lógicas. La primera sería convertir en campos clave en nombre y el sabor (claro caso de campo combinado), ya que estos si que no se deben repetir nunca. No podemos tener en un registro 300 Calipos de sabor fresa y en otro 130 Calipos más del mismo sabor, ya que en este caso se sumarían, y tendríamos 430 Calipos de sabor fresa, pero en un solo registro. La otra posibilidad es la de añadir un nuevo campo, Llamado Id (identificador), que actúe como campo clave, y se incremente progresivamente por cada registro que añadamos (claro caso de campo clave auto-numérico)
.
Vamos al Control Center de MySql y buscamos la tabla stock. La editamos y creamos el nuevo campo clave "Id", indicando que será AUTO_INCREMENT (auto incremental).

Añadamos también la correspondiente columna a la lista y agreguémosla en el código.

```
With Lista.ListItems.Add(, , Rs("Id"))
        .SubItems(1) = Rs("Nombre")
        .SubItems(2) = Rs("Sabor")
        .SubItems(3) = Rs("Cantidad")
    End With
```

Bien, lo que vamos a hacer ahora es crear dos procedimientos que habiliten y deshabiliten los botones Aceptar Cancelar, así como las cajas de texto.

```
Sub Deshabilitar()
  CmdAceptar.Enabled = False
  CmdCancelar.Enabled = False
  Txtnombre.Enabled = False
  TxtSabor.Enabled = False
  TxtCantidad.Enabled = False
End Sub
Sub Habilitar()
  CmdAceptar.Enabled = True
  CmdCancelar.Enabled = True
  Txtnombre.Enabled = True
  TxtSabor.Enabled = True
  TxtCantidad.Enabled = True
End Sub
```

También vamos a hacer otro procedimiento que nos moverá los datos del "Item" seleccionado en la lista a las cajas de texto.

```
Sub MoverDatosACampos()
    Txtnombre = Lista.SelectedItem.SubItems(1)
    TxtSabor = Lista.SelectedItem.SubItems(2)
    TxtCantidad = Lista.SelectedItem.SubItems(3)
End Sub
```

Estos dos métodos serán llamados, para empezar, después de la primera carga en el evento "Form_Load".

```
   MoverDatosACampos
  Deshabilitar
End Sub
```

(Este "End Sub" pertenece al Form_Load) Si todo ha ido bien, ahora debería cargarse los campos con los datos del "Item" de la lista al picar en el botón "Stock" del MDI. También llamaremos a este procedimiento desde el evento doble click del ListView

```
Private Sub Lista_DblClick()
  MoverDatosACampos
End Sub
```

Ahora deberían cargarse las cajas de texto al efectuar doble click en algún "Item" de la lista.

Perfecto, lancémonos entonces a por el primer mantenimiento, por ejemplo, el botón modificar. Lo primero que vamos a hacer es una función que compruebe que los campos tienen valores totalmente válidos, ya que de lo contrario tendríamos un error. Por ejemplo, yo la he llamado "CamposOK", que devolverá verdadero si son correctos y falso si no lo son.

```
Function CamposOK() As Boolean
  If Txtnombre.Text = "" Then
    MsgBox "Debe rellenar el campo nombre", vbExclamation, "Aviso"
    CamposOK = False
    Exit Function
  End If
  If Not IsNumeric(TxtCantidad) Then
    MsgBox "El campo cantidad debe ser numérico", vbExclamation, "Aviso"
    CamposOK = False
    Exit Function
  End If
  CamposOK = True
End Function
```

Vemos que se trata de una función y no un procedimiento porque debe devolver un valor indicándonos si los campos son correctos o no. También podemos apreciar que no comprobamos el valor del campo Sabor, ya que lo hemos declarado como que acepta valores nulos, porque puede que un helado no tenga un sabor específico o sea único en su gama.



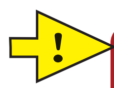
La función...

La función "IsNumeric" devuelve verdadero cuando el valor de lo que introducimos entre los paréntesis es un número.

Para ser más pulcros, también codificaremos la caja de texto "Cantidad" para que solo acepte números. Por ejemplo, podríamos borrar el contenido de esta cuando el usuario, inteligente él como nadie, intente introducir letras en un campo cuyo nombre es "Cantidad" (aunque parezca ilógico, "ellos" son capaces de hacerlo)

```
Private Sub TxtCantidad_KeyUp(KeyCode As Integer, Shift As Integer)
    If Not IsNumeric(TxtCantidad) Or KeyCode = 46 Or KeyCode = 44 Then
        TxtCantidad = 1
    End If
End Sub
```

Le estamos diciendo que, en el caso de que se introduzca un valor no numérico, o se introduzca un punto o una coma, el contenido de la caja de texto pase a ser "1".



La variable...

La variable KeyCode contiene el código ASCII de la tecla pulsada cuando el foco está en la caja de texto.

Vamos ahora al botón Modificar. Para saber que opción hemos pulsado nos declararemos una variable que variará dependiendo si hemos picado en Modificar, Nuevo o Borrar. La llamaremos "Estado"

```
Option Explicit
Dim Estado As String
```

Desde el evento Click del botón modificar únicamente cargaremos esta variable con su correspondiente valor, habilitaremos los botones Aceptar Cancelar y deshabilitaremos los botones Nuevo, Modificar y Borrar. Este sería el código, rutinas incluidas

```
Private Sub CmdModificar_Click()
    Habilitar
    Estado = "MODIFICAR"
    DeshabilitarBotones
End Sub


Sub DeshabilitarBotones()
    CmdModificar.Enabled = False
    CmdBorrar.Enabled = False
    CmdNuevo.Enabled = False
End Sub

Sub HabilitarBotones()
    CmdModificar.Enabled = True
    CmdBorrar.Enabled = True
    CmdNuevo.Enabled = True
End Sub
```

En el botón Aceptar comprobaremos el valor de la variable "Estado", y dependiendo del resultado, modificaremos, daremos de alta o borraremos. Para modificar, debemos posicionarnos al principio del Recordset, buscar el registro que deseamos modificar por el campo clave y mover los datos al registro. Este es un posible código

```
Private Sub CmdAceptar_Click()
    Select Case Estado
        Case "MODIFICAR"
            If CamposOK = True Then
                Rs.MoveFirst
                Rs.Find "Id=" & Lista.SelectedItem.Text
                If Not Rs.EOF Then
                    MoverDatosARegistro
                    Rs.Update
                    MsgBox "Registro modificado"
                    Lista.ListItems.Clear
                    Call Form_Load
                Else
                    MsgBox "No se ha encontrado el registro seleccionado, se
                    va a recargar la lista", vbCritical, "Error"
                    call Form_Load
                End If
            End If
    End Select
    HabilitarBotones
    Deshabilitar
End Sub

Sub MoverDatosARegistro()
    Rs("Nombre") = Txtnombre
    Rs("Sabor") = TxtSabor
    Rs("Cantidad") = TxtCantidad
End Sub
```

Vemos que después del mensaje "Registro modificado", escribimos "Call Form_Load". Esta es la manera que tiene Visual Basic para llamar a un evento. En este caso, para recargar la lista, llamamos al evento Form_Load, pero bien podríamos haber codificado la carga de la lista en una rutina y llamarla en este momento.

El mantenimiento "nuevo" es prácticamente igual que el de modificación, con la diferencia que deberemos hacer un "AddNew" antes de pasar los datos a los campos.
El código del botón "Nuevo" sería este:

```
Private Sub CmdNuevo_Click()
    Habilitar
    Estado = "NUEVO"
    DeshabilitarBotones
    LimpiarCampos
End Sub

Sub LimpiarCampos()
    Txtnombre.Text = ""
    TxtSabor.Text = ""
    TxtCantidad.Text = 1
End Sub
```

Si le echamos un vistazo, estamos haciendo lo mismo que en botón "Modificar" con la diferencia que borramos el contenido de las cajas de texto. Ahora, codificamos en el botón "Aceptar" para que haga las operaciones necesarias al intentar dar de alta.

```
        Case "NUEVO"
         If CamposOK = True Then
            Rs.AddNew
            MoverDatosARegistro
            Rs.Update
            MsgBox "Registro dado de alta"
            Lista.ListItems.Clear
            Call Form_Load
         End If
```

También muy parecido, pero con la diferencia que en vez de buscar un registro para modificar, hacemos un "AddNew" para añadir uno nuevo.

He dejado en estas líneas lo que yo consideraría un "gazapo"(no es un error de código), que si habéis leído todo el artículo, lo tendríais que ver. Posteriormente lo explicaré, pero me gustaría que vierais si es lógico lo que estamos haciendo. Y para acabar, el botón "Borrar"

```
Private Sub CmdBorrar_Click()
    CmdAceptar.Enabled = True
    CmdCancelar.Enabled = True
    Estado = "BORRAR"
    DeshabilitarBotones
End Sub
```

No llamamos a la función "Habilitar" porque es innecesario habilitar las cajas de texto.
Para borrar, en el botón aceptar buscaremos el registro seleccionado y lo eliminaremos con el método "Delete".

```
        Case "BORRAR"
          Rs.MoveFirst
          Rs.Find "Id=" & Lista.SelectedItem.Text
          If Not Rs.EOF Then
             Rs.Delete
             MsgBox "Registro borrado"
             Lista.ListItems.Clear
             Call Form_Load
          Else
             MsgBox "No se ha encontrado el registro seleccionado, se va a recargar la lista", vbCritical, "Error"
             Call Form_Load
          End If
```

Como veis, tanto Modificar, Borrar y Nuevo son muy parecidos. Tal vez la diferencia más visible es que cuando queremos hacer algo contra un registro en concreto (Modificar o Eliminar), antes debemos posicionarnos, por ejemplo, con el método "Find", mientras que cuando queremos dar de alta no nos hacer falta. Y que no se me olvide, los botones "Cancelar" y "Salir"

```
Private Sub CmdCancelar_Click()
    Deshabilitar
    HabilitarBotones
End Sub

Private Sub CmdSalir_Click()
    Unload Me
End Sub
```

Bueno, pues con esto hemos acabado por hoy. Este formulario ha quedado altamente mejorable, así que poneros manos a la obra. En la próxima entrega seguiremos con el resto, que aunque serán parecidos, haremos una introducción al SQL. También podréis tener la gratificación de haber hecho un proyecto medianamente importante, que bien podréis modificar a vuestro antojo e intentar generalizar para que funcionen en varios tipos de negocio. Un saludo, y nos vemos!!! --Si no te gusta teclear, en la Web tienes el código completo--



GAZAPO

Gazapo: Pues si nos fijamos en el código para dar de alta, nos damos cuenta que ignora totalmente si ya existe un helado con ese nombre y ese sabor. Debería sumar las cantidades, es decir, modificar el registro original sumándole el contenido de TxtCantidad, mientras que ahora, erróneamente, está añadiendo un nuevo registro. ¿Os veis capaces de hacerlo sin mi ayuda? ---en la próxima entrega os daré la solución ;) ---

RAW 7 : HTTP
(HYPERTEXT TRANSFER PROTOCOL)

De nuevo nos adentramos en el conocimiento de los protocolos, esta vez le toca el turno al HTTP, para que nos entendamos, el que utilizamos al visualizar una Web :)

1. Un poco de historia

Cuando en el año 1965 Ted Nelson acuñó el término "hipertexto", difícilmente podría imaginar la extensión que llegaría a alcanzar su idea. En realidad, aunque la idea de crear un sistema de texto con enlaces es muy anterior, no fue hasta 1989-1990 cuando Tim Berners-Lee, informático del CERN (Organización Europa de Investigación Nuclear), desarrolló el primer sistema web para que los científicos del CERN pudieran compartir información. El navegador que utilizaban estos científicos para acceder al sistema fue precisamente el primer navegador web de la historia, se llamaba "WorldWideWeb" (posteriormente llamado "Nexus").

Aun hoy podemos ver una copia de esta primera página tal y como era en el año 1992 (no hay una copia de la página en su primera aparición) en la dirección:
http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html.

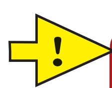
Quizá nos pueda parecer ahora algo totalmente cotidiano, e incluso increíblemente simple, el concepto del hipertexto y la www, pero hay que imaginarse la visión que se podría tener de estas cosas hace 40 años, cuando sólo pertenecían a la ciencia-ficción. Sin irme tan lejos, yo aún recuerdo cuando vi por primera vez la película BIG, en el año 88 (poco antes de que comenzase todo), y me quedé maravillado con la idea de que pudiese existir un libro electrónico con el que pudieses interactuar escogiendo diversos finales, etc. (algo así como los "Elige tu propia aventura" pero en plan futurista).

A pesar de que el primer navegador fue gráfico, éste sólo existía para plataforma NeXT, por lo que los usuarios que tuviesen otros sistemas tenían que conformarse con un simple navegador en modo texto. No fue hasta 1993 cuando apareció la primera versión del antes popular navegador "Mosaic" para X-Windows, que era quizá por aquel entonces la plataforma más extendida entre los usuarios de Internet (formada en su mayor parte por científicos y algunos estudiantes). A partir de la aparición de Mosaic (que pocos meses después también estaba disponible para PC y Macintosh) comenzó la gran explosión de la WWW, con un crecimiento exponencial de vértigo.

Mi primer contacto con la WWW fue en el año 1995 y, a pesar de que ya existían navegadores gráficos, di mis primeros pasos con el navegador en modo texto "Lynx", en un sistema VMS (uno de esos muchos sistemas operativos que suenan a chino para la mayoría). No fue hasta el año siguiente cuando entré en contacto por primera vez con un navegador gráfico, Netscape, al conectar por primera vez desde mi casa con un modem último modelo de 14400, y el entonces popular Trumpet Winsock. Recuerdo también cómo hice entonces mi primera página web, escribiendo código HTML a pelo en el bloc de notas de Windows... Que tiempos!!! XDDD

2. Documentación

No se puede hablar del protocolo **HTTP** sin hablar de otras cosas íntimamente ligadas, como son el lenguaje **HTML**, el estándar **MIME**, etc. Si bien la especificación del protocolo propiamente dicho se encuentra en el **RFC 2616** (ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt), para tener una documentación completa sobre el tema no basta con centrarse en los RFCs, si no que hay que documentarse también adecuadamente sobre estos otros aspectos.



Para los nuevos...

Para los nuevos lectores, simplemente recordarles que hemos explicado ampliamente en números anteriores **qué son los RFC**. En http://www.rfc-es.org encontrarás los RFC que han sido traducidos al Español, si el inglés no es lo tuyo, esta Web es de visita obligada. Por cierto, si eres un portento en esto de las traducciones puedes aportar tu granito de arena ofreciéndote a traducir alguno de los muchos RFC que todavía no han sido traducidos :)

Podéis encontrar en Google o en cualquier librería miles de tutoriales de **HTML**.

Por poner un ejemplo, tenéis uno en: http://www.davesite.com/webstation/html/ (lo único que he hecho para encontrar esta página ha sido pinchar en el primer resultado que me daba Google al buscar "html tutorial" --> http://www.google.com <--).

Con respecto al **MIME**, se encuentra también definido en varios RFCs. En la página: http://www.mhonarc.org/~ehood/MIME/ podéis encontrar una lista de RFCs que especifican éste estándar. Si no queréis leeros los 5 capítulos completos (**RFC 2045, RFC 2046, RFC 2047, RFC 2048, y RFC 2049**) podéis haceros una idea básica leyendo el ya obsoleto **RFC 1521** (ftp://ftp.rfc-editor.org/in-notes/rfc1521.txt).



En el número 12...

En el número 12 de PC PASO A PASO recomendamos la lectura de un artículo que, a pesar de su antigüedad, explica de forma muy amena términos como MIME, ASCII, ISO, UUENCODE, UUDECODE, BASE64... No dejes de leerlo, puedes encontrarlo en http://bitassa.com/articles/babel.html

Pero empecemos aclarando un poco este barullo. ¿Qué relación guardan entre sí el HTTP, el HTML, y el MIME?

El **HTTP** (**H**yper**T**ext **T**ransfer **P**rotocol) no es más que un protocolo que funciona sobre TCP/IP. Al igual que los demás protocolos que hemos ido viendo a lo largo de la serie RAW, el protocolo HTTP está formado por una serie de comandos y una serie de respuestas que el servidor proporciona a esos comandos. Los comandos son muy pocos, pero la **complicación del protocolo se encuentra en los llamados campos de cabecera** que son, en cierto modo, como los parámetros que se pasan con cada comando y con cada respuesta.

El **HTML** (**H**yper**T**ext **M**ark-up **L**anguage), como supongo que todos sabréis, es un lenguaje de representación de hipertexto. Podríamos decir que el HTTP es como un FTP muy simple pero limitado a transferir sólo un tipo de contenidos. En FTP puedes transferir cualquier tipo de archivos, y en HTTP transfieres básicamente sólo código HTML.

Con respecto al **MIME** (**M**ultipurpose **I**nternet **M**ail **E**xtensions), hay que tener en cuenta que el lenguaje HTML permite la inclusión de cualquier tipo de contenidos (texto en diferentes idiomas, incluidos algunos como el japonés que no utilizan los mismos caracteres que el inglés, imágenes en diferentes formatos, audio, vídeo, etc., etc.). El MIME es un estándar para la representación de estos contenidos, que no sólo es usado en la WWW, si no también en

cualquier entorno de Internet que sea susceptible de transportar diferentes contenidos. De hecho, si repasáis los números anteriores de la serie RAW, veréis que ya comenté algo acerca del MIME en los artículos sobre POP3 y SMTP. HTTP y MIME no son totalmente compatibles. En el **apéndice 19.4** del **RFC 2616** podéis encontrar detalladas las diferencias entre ambos estándares.

Como podéis suponer, con este artículo os ahorraré el leer toda esta documentación si lo único que queréis es tener una visión global sobre el tema, y no queréis convertiros en los nuevos gurus de la www. :-D

Me centraré tan sólo en el protocolo HTTP, ya que el MIME y el HTML merecerían artículos aparte, aunque no de la serie RAW, ya que no son protocolos. ;-)

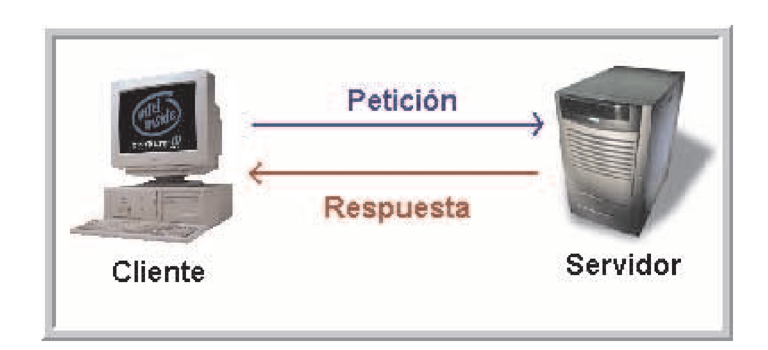
3. Arquitectura HTTP

El protocolo HTTP ha sufrido algunas modificaciones desde su aparición, y actualmente se encuentra en su versión **1.1**.

* La primera versión, **HTTP/ 0.9**, era un protocolo muy simple para la transmisión de datos sin formato, poco más que un FTP simplificado.
* La versión **HTTP/ 1.0** (**RFC 1945**) soportaba ya los contenidos MIME, pero no estaba preparada para soportar algunas características avanzadas de la www, como la cache, los virtual hosts, etc.

* La actual versión **HTTP/ 1.1** surge de la necesidad de dar soporte a estas características. En el **apéndice 19.6.1** del **RFC 2616** se detallan las diferencias entre **HTTP/ 1.0** y **HTTP/ 1.1**.

Una conexión básica de HTTP consiste en una petición de un cliente, y una respuesta del servidor. Típicamente, esta petición suele ser una URL, y la respuesta suele ser el contenido HTML de esa página web.





Para los lectores novatos

Cada vez que abres el Internet Explorer (o cualquier otro navegador de Internet) y le metes una dirección Web (por ejemplo www.mocosoft.com) estás estableciendo una conexión básica de HTTP. Tu eres el CLIENTE (el que pide) y el SERVIDOR WEB (el que sirve) donde está alojada la página www.mocosoft.com responderá a tu petición enviándote (sirviéndote) el contenido HTML de la página. Tu "querido" Internet Explorer interpretará ese código HTML y mostrará el resultado en tu monitor.

En cambio, la arquitectura puede ser más compleja (uno de los motivos por los cuales se hizo conveniente la aparición de HTTP/ 1.1), incluyendo varias máquinas intermediarias en la comunicación. Estos intermediarios pueden ser básicamente de 3 tipos: proxies, gateways, y túneles.

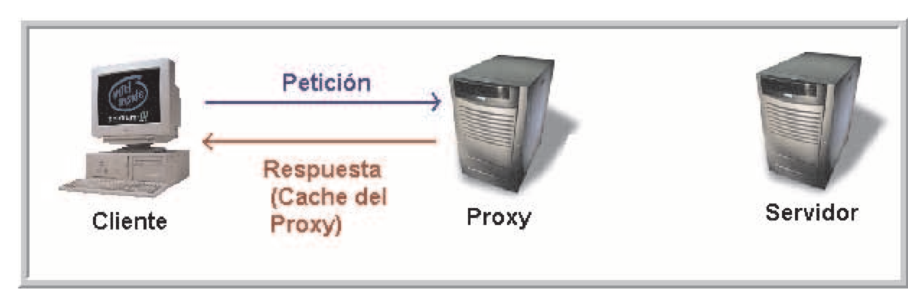
Un **túnel** es simplemente un intermediario que retransmite todo lo que recibe sin siquiera tener que comprender ninguno de los contenidos que circulan a través de él. Un **gateway**, en cambio, puede realizar traducciones de protocolos cuando las arquitecturas a ambos lados son incompatibles, pero los contenidos no se verán afectados y las traducciones serán transparentes para ambas máquinas. Por último, un **proxy**, puede modificar los datos transferidos, modificando, por ejemplo, las cabeceras de las

peticiones (¿recuerdas cuando explicamos en números anteriores a navegar anónimamente mediante proxy? ;p)

Petición
Petición
Respuesta
Respuesta
Cliente
Proxy
Servidor

Otra de las características avanzadas de la arquitectura que permite manejar el protocolo HTTP/ 1.1 es la **cache**. En el escenario anterior, en el que un cliente conecta con un servidor a través de varias máquinas intermedias, puede que la respuesta que reciba el cliente no provenga directamente del servidor final, aunque esto habrá de ser transparente para el usuario. Alguna de las máquinas intermedias podría tener una copia de la respuesta del servidor ante esa petición, y enviarla directamente al cliente sin necesidad de recorrer de nuevo todo el camino de ida y de vuelta hasta el servidor final.

Normalmente, las máquinas intermedias tendrán en cache las respuestas que ya han sido solicitadas previamente por otros clientes (o incluso por ese mismo cliente), aunque pueden incluso tener copias de cache de las páginas Más solicitadas distribuidas en CD.



El tema de la cache en HTTP es realmente complejo, y podría duplicar la extensión de este artículo, por lo que lo obviaré, al no ser necesario para una comprensión básica del protocolo.

4. Niveles de conexión

Hacer todo el estudio del protocolo HTTP con un cliente de Telnet puede ser duro, ya que a veces nos bastará una herramienta que abstraiga algunos parámetros que no nos interesen.

Todos conocemos la herramienta principal de más alto nivel de la que disponemos para manejar el protocolo HTTP: los **navegadores** (browsers). Existe una cantidad enorme de navegadores para todos los sistemas y arquitecturas, y no sólo Internet Explorer (IE), como parece que muchos quieren hacernos creer.

Ya que menciono IE, os recuerdo que es una de las aplicaciones más inseguras que existen hoy día. Para que comprobéis que mi afirmación no es gratuita, os propongo que hagáis el experimento de visitar las más importantes páginas de seguridad informática, y realizar una búsqueda de la cadena "Internet Explorer". Os pongo como ejemplo una lista de páginas de seguridad que he consultado, y el número de avisos de seguridad que aparecen en esa página sobre IE:

- www.securityfocus.com (569 coincidencias)
- www.securitytracker.com (301 coincidencias)
- www.cert.org (1538 coincidencias)
- www.theregister.co.uk (447 coincidencias)
- www.hispasec.com (superado el límite de 50 coincidencias)
- neworder.box.sk (superado el límite de 20 coincidencias en la sección de exploits, el límite de 30 coincidencias en la sección de exploits antiguos, el límite de 20 coincidencias en la sección de artículos, y el límite de 20 coincidencias en la sección de noticias breves)

Por supuesto, entre todas esas entradas, alguna se habrá colado que no sea un aviso de seguridad, pero podéis realizar el experimento por vosotros mismos y comprobar que la inmensa mayoría de las coincidencias son referentes a la "seguridad" de IE.

Tenéis muchas alternativas a IE en sistemas Windows, y no solo Netscape, como también intentan hacernos creer (de hecho, Netscape será quizá más seguro que IE, pero por mi experiencia personal puedo afirmar que funciona realmente mal). Yo personalmente, en Windows me quedo con el navegador **Opera**. La versión shareware (gratis, para que nos entendamos) es totalmente funcional y simplemente añade un banner de publicidad en una de las esquinas. En realidad se puede utilizar perfectamente con el banner, que molesta poco, pero si queréis quitarlo no tenéis más que pagar el módico precio del registro. Porque claro, doy por hecho que estoy hablando con gente legal, y que no se os ocurrirá visitar http://astalavista.box.sk para buscar un serial number de Opera y registrarlo gratuitamente... :p

En Linux uso también Opera, por costumbre más que nada, pero tenéis también un amplísimo repertorio de navegadores (Mozilla, Konqueror, Galeon, Netscape, etc).

Centrándonos de nuevo en el tema, si queremos utilizar un navegador (sea el que sea) como herramienta HTTP, tenemos que conocer perfectamente el formato de una URL (Uniform Resource Locator), que es el siguiente:

http://host[:puerto][/path][?consulta]

* El campo host es el único obligatorio, y es donde ponemos la dirección del servidor HTTP. Podemos poner bien una dirección **IP** (217.174.193.62), o bien un nombre **DNS** (www.hackxcrack.com).
* El campo puerto normalmente no será necesario, ya que el puerto por defecto para HTTP es el **80**. En cambio, si es necesario, podemos especificar cualquier otro puerto.
* El campo path es un nombre de directorio relativo al directorio configurado como raíz para el servidor web. Esto es totalmente transparente al usuario, por lo que desde el punto de vista del cliente se puede decir que se trata de un directorio absoluto. En realidad, como ya sabemos, existen formas de saltarse esto del directorio raíz del servidor web, y acceder a cualquier otro directorio de la máquina. ;-)
* El campo consulta es opcional, y es sólo para páginas de contenido dinámico (PHP, ASP, CGI, ...) en las que se puedan introducir parámetros.

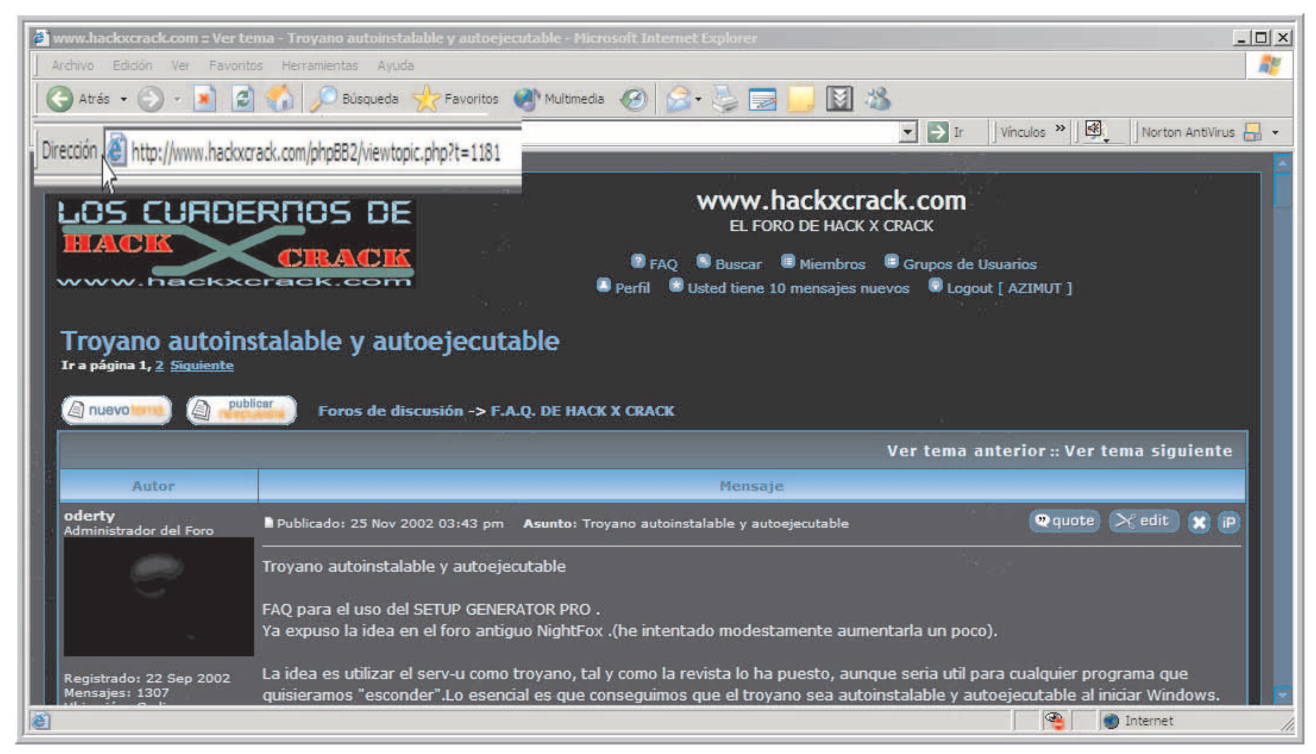


A partir de ahora...

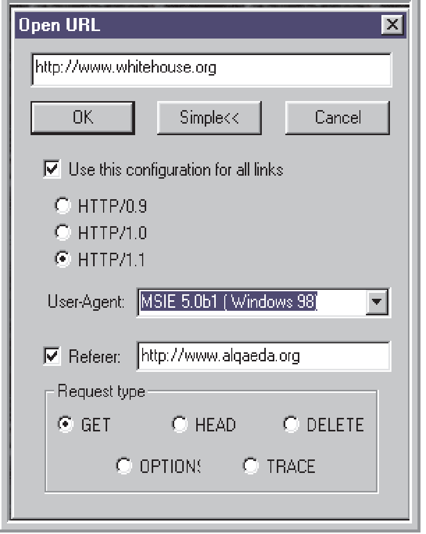
A partir de ahora, si eres de los que poseen una mente inquieta, te recomendamos que cuando visites una página Web no pierdas de vista el campo dirección de tu navegador. Podrás ver "en directo" lo "interesantes" que pueden ser algunas peticiones(vease imagen siguiente)

Por otro lado, recuerda que cuando introduces en tu Navegador una dirección Web (por ejemplo www.astalavista.com), el Navegador intenta por defecto acceder al puerto 80. No tienes mas que probarlo, escribe por ejemplo http://www.astalavista.con:80 en tu navegador y verás que el resultado es el mismo que si escribieses http://www.astalavista.com. Lo que no sabe todo el mundo es la cantidad de Servidores Web que hay en Internet tras otros puertos, si has seguido el curso del Servidor Web APACHE (en anteriores números de PC PASO A PASO) sabes que puedes activar un Servidor Web en cualquier puerto, por ejemplo en el puerto 4415… no te imaginas la cantidad de Servidores Web que en el Puerto 80 tienen páginas Web de lo más normalitas y en "otros puertos" tienen las páginas interesantes ;p

Si www.astalavista.com tuviese una web oculta en el puerto 9010, para acceder a ella tendrías que introducir en tu navegador www.astalavista.com:9010. Ya, ahora me pedirás una lista de Webs "ocultas", pues lo siento pero eso sería suficiente para que esta revista fuese repudiada en ciertos círculos, sería como publicar el acceso a uno de los muchos FTP de los que utiliza la scene… si quieres adentrarte en este tema, empieza por trabajarte los foros de FXP ;) … ¿Cómo? ¿Qué no sabes lo que es eso? Pues repasa los anteriores números de PC PASO A PASO ;p



La combinación de un navegador de los de toda la vida, y un **sniffer**, puede ser una herramienta muy útil para estudiar el funcionamiento del protocolo HTTP. Existen también herramientas intermedias entre esto y un simple cliente de Telnet, como por ejemplo la aplicación **SamSpade**, que no sólo puede ser descargada como aplicación para usar localmente en tu PC, si no que también puede ser utilizada online desde el propio servidor de SamSpade: www.samspade.org.



En la imagen podemos ver la configuración de la herramienta Browse Web incluida en SamSpade, en la que configuramos la aplicación para que se conecte a www.withehouse.org haciéndole creer que nuestro navegador es un Internet Explorer 5.0 (cuando en realidad nuestro cliente es SamSpade), y que la página de la que venimos es www.alqaeda.org. En el parámetro Request Type podemos especificar el comando HTTP a enviar. Más adelante veremos cada uno de estos comandos en detalle.

Por último, como no, hay que mencionar el cliente de **Telnet** como aplicación universal para el estudio de los protocolos. ;-) Si utilizamos el cliente de Telnet tenemos que tener en cuenta dos aspectos. El primero, es que el puerto por defecto para HTTP es el **80**. Y por último, que no podemos incluir el path al lanzar la conexión mediante Telnet, si no sólo el host. Para incluir un path, tendremos que hacerlo una vez establecida la conexión con el servidor, enviando la cabecera adecuada, tal y como veremos más adelante.

5. El protocolo HTTP

Una vez establecida la conexión (asumimos que estamos trabajando con un cliente de Telnet), el servidor se queda en espera de que enviemos un comando, al cual responderá después tal y como iremos viendo.

Teniendo en cuenta que el **RFC 2616**, que especifica el protocolo **HTTP /1.1**, consta de 175 páginas, no podemos hacer en este artículo una especificación detallada y precisa del protocolo, por lo que nos limitaremos a ver casos típicos y relativamente sencillos.

En primer lugar, tenemos que tener en cuenta que no todos los servidores aceptan todos los comandos HTTP. Cuando un servidor no acepte un determinado comando, nos responderá con un **error 405 (method not allowed)**. Si ni siquiera conoce ese comando, nos responderá con un **error 501 (Not Implemented)**. Sólo los comandos **GET** y **HEAD** han de ser implementados obligatoriamente por cualquier servidor.

El protocolo HTTP consta de muy pocos comandos RAW (tan sólo 8, de los cuales sólo hay 2 cuya implementación es obligatoria). Toda la complejidad del protocolo no se encuentra en los propios comandos, si no en los parámetros adicionales que se envían con cada petición y con cada respuesta. El número de estos parámetros varía de una petición a otra, y se encuentra cada parámetro separado en una línea diferente. A cada una de estas líneas se las denomina **campos de cabecera (header fileds).** Cuando un campo de cabecera no se especifica explícitamente, el estándar define una serie de decisiones a tomar por defecto tanto por parte del servidor como por parte del cliente, por lo que no existe un número fijo de campos de cabecera obligatorios.

Para tratar de explicaros de forma ordenada todo este barullo que se encuentra repartido por varios documentos diferentes, y de forma bastante caótica, he decidido empezar mostrando un ejemplo básico de sesión completa, capturada con un **sniffer**, para luego ir explicándola paso a paso. Para esta explicación empezaré por comentar los 8 comandos RAW, para luego hablar de los campos de cabecera. Por último, hablaré sobre as respuestas del servidor.



Para los nuevos lectores

¿TELNET? ¿SNIFFER? ¿Qué es eso? ¿Cómo se utiliza? En los números anteriores mostramos paso a paso como utilizar estas herramientas, si es la primera vez que lees esta revista y no dominas el tema seguro que ya te has perdido. La solución ideal pasa por comprar los números atrasados de PC PASO A PASO, claro, pero estamos estudiando una alternativa que esperamos poder poner en marcha a mediados de Noviembre (máximo Diciembre).

Lo que tenemos pensado es hacer en una especie de FAQ y colgarla en la Web (www.hackxcrack.com), en realidad será un archivo que todo el mundo podrá descargar y contendrá esos artículos ya publicados y de nivel básico **que todo nuevo lector deberá "tener muy a mano"**. No vamos a liberar los PDF de todos los números publicados, será simplemente una recopilación de todo lo que creemos imprescindible para que cualquier nuevo lector pueda seguir la revista.

Empiezo, por tanto, mostrándoos la sesión completa de http. Os recuerdo que a lo largo de toda la serie RAW he utilizado el color azul para indicar lo que envía el cliente al servidor, y el color rojo para las respuestas del servidor al cliente. El color verde lo utilizo para comandos de consola, enviados fuera de la conexión TCP/IP.

```
GET / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; UNIX) Opera 6.11  [en]
Host: www.hackxcrack.com
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*
Accept-Charset: windows-1252, utf-8;q=1.0, utf-16;q=1.0, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Cookie: phpbb2mysql_data=a%3B1%5B%5etc
Cache-Control: no-cache
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers

HTTP/1.1 200 OK
Date: Thu, 09 Oct 2003 14:59:29 GMT
Server: Apache/1.3.26 (Unix) PHP/4.2.3 mod_perl/1.26
Last-Modified: Fri, 23 May 2003 19:52:13 GMT
```

```
Accept-Ranges: bytes
Content-Length: 6291
Content-Type: text/html
Age: 0
```

...AQUI VENDRIA UN MONTON DE CODIGO HTML...

```
GET /imagenes/LOGOPCPASOAPASO1.jpg HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; UNIX) Opera 6.11 [en]
Host: www.hackxcrack.com
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*
Accept-Charset: windows-1252, utf-8;q=1.0, utf-16;q=1.0, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://www.hackxcrack.com/
If-Modified-Since: Thu, 03 Apr 2003 06:06:09 GMT
If-None-Match: "710029-10d4-3e8bcf51"
Cookie: phpbb2mysql_data=a%3A2%3A%etc
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

```
HTTP/1.1 304 Not Modified
Date: Thu, 09 Oct 2003 14:59:30 GMT
Etag: "710029-10d4-3e8bcf51"
```

¡¡Buf!! ¡Como podéis imaginar, tenemos mucho trabajo por delante! ;)

5.1. Comandos RAW de HTTP

Enumeraré a continuación todos los comandos HTTP, aunque nos centraremos sobre todo en el comando **GET**, que es el más importante. Mientras explique los comandos, probablemente habrá muchas cosas que os suenen a chino. No os preocupéis, porque más adelante explicaré todo en detalle.

5.1.1. Comando GET

Este es, sin duda, el comando HTTP más utilizado, ya que es el que se usa cuando queremos descargar una página web de un servidor. Vamos a ver dos formas diferentes de conectarnos a la URL http://www.hackxcrack.com/entrada/entrada.htm mediante Telnet:

Conexión directa:

Lo primero será lanzar el telnet al servidor:

```
telnet www.hackxcrack.com 80
```

A continuación, el comando sería:

```
GET /entrada/entrada.htm HTTP/1.1
```

Como vemos, indicamos sólo el PATH, separado por un espacio del protocolo a utilizar que, en este caso, y prácticamente en cualquier caso, será siempre HTTP/1.1.

Después de esta línea, debemos indicar aparte el HOST para completar la URL:

```
GET /entrada/entrada.htm HTTP/1.1
Host: www.hackxcrack.com
```

Esto permite referenciar otros HOSTs a partir de uno dado, siempre que éste lo permita. Si el host especificado no es válido, el servidor responderá con un **error 400 (Bad Request).** Este campo **Host** que hemos visto que sigue al propio comando es precisamente uno de los campos de cabecera que mencioné. En el ejemplo de sesión completa que puse al principio podemos ver lo siguiente:

```
GET / HTTP/1.1
Host: www.hackxcrack.com
```

En este caso, la página solicitada es la raíz del servidor.

Conexión a través de un proxy:

Si conectasemos a través de un proxy, por ejemplo 127.0.0.1, tendríamos que especificar no sólo el PATH, si no también el HOST como parámetro del comando **GET**. Utilizar esta IP como proxy no es ninguna tontería (para los que se puedan quejar de que no ponemos IPs reales en los artículos), ya que podríamos estar utilizando una aplicación como **MultiProxy** (www.multiproxy.org –utilidad explicada en números anteriores-) que crea un proxy virtual en nuestro propio PC que lo que hace es manejar dinámicamente listas de proxies externos.

Si tenemos, por ejemplo, un multiproxy en el puerto 8080 de nuestra máquina, empezaremos con el siguiente Telnet:

telnet 127.0.0.1 8080

A continuación, lanzaremos la siguiente petición:

GET http://www.hackxcrack.com/entrada/entrada.htm HTTP/1.1

** No he probado este ejemplo concretamente con Multiproxy. Lo único que comento es el método genérico para conectar a través de un proxy.

5.1.2. Comando OPTIONS

Este comando nos permite conocer las características de la comunicación entre nuestro cliente y un recurso determinado del servidor. Si, por ejemplo, enviamos:

OPTIONS * HTTP/1.1

El recurso solicitado es *, que es equivalente a no especificar ningún recurso. Esto equivale a hacer un simple PING al servidor para ver si responde.

Si queremos enviar un comando **OPTIONS** a un proxy en concreto dentro de una cadena, en lugar de enviarlo al servidor final al que supuestamente estamos conectados, podemos utilizar el campo de cabecera **Max-Forwards**, tal y como veremos más adelante.

5.1.3. Comando HEAD

Es similar al comando **GET**, pero con la diferencia de que el servidor en su respuesta no incluirá los datos a enviar (típicamente el código HTML), si no tan sólo la cabecera. Con esto se puede comprobar si una URL es válida, cuál es el tipo de contenidos, si ha de ser renovada la copia de cache debido a algún cambio, etc.

5.1.4. Comando POST

Este comando permite que el cliente solicite el envío de datos al servidor, en lugar de solicitar una recepción de datos. Es, por ejemplo, el método utilizado para subir un mensaje a un foro web. Si el mensaje ha sido añadido con éxito al foro, el servidor nos responderá con un **código 201 (Created)**.

El comando **POST** debe incluir, por supuesto, un cuerpo de datos con su cabecera, igual que hace el servidor cuando nos envía una página solicitada. Más adelante veremos cómo construye el servidor estas cabeceras.

5.1.5. Comando PUT

Este comando es el inverso a **GET**, ya que permite especificar una URL sobre la que se escribirán los datos incluidos en la propia petición. Si no existía previamente ese recurso, el servidor nos responderá con un **código 201 (Created)**, en cambio, si ya existía y ha sido modificado, nos responderá con un **código 200 (Ok)**, o un **código 204 (No Content)**. Si el servidor no comprende los contenidos enviados, responderá con un **error 501 (Not Implemented)**.

Lo importante que hay que comprender en este punto es la diferencia entre los comandos **POST** y **PUT**. En el comando **POST**, la URL especificada es la de algún recurso que sea capaz de procesar los datos, por ejemplo, un script CGI, PHP, o ASP. Este script decidirá qué hacer con los datos enviados. En cambio, en el comando **PUT** la URL especificada será directamente sobre la que se escribirá.

5.1.6. Comando DELETE

Como es fácil adivinar, sirve para eliminar el recurso especificado en la URL. Por supuesto, este comando no funcionará con la inmensa mayoría de URLs. XDD

5.1.7. Comando TRACE

Es prácticamente como un PING, no sólo hacia el servidor final, si no también hacia alguno de los proxies intermedios si se utiliza el campo de cabecera **Max-Fordwards**. El servidor que haya de responder, lo hará con un **código 200 (Ok)**, y la respuesta tendrá como tipo de

contenidos **message/http**. Esta respuesta puede ser utilizada por el cliente para hacer diversos diagnósticos.

5.1.8. Comando CONNECT

Este comando se utiliza en los proxies que pueden funcionar como tunel SSL. Esto es un tema muy complicado, así que mejor lo dejamos, que no es plan de escribir un libro. xD

5.2. Campos de cabecera

La petición se puede completar con una serie de campos adicionales, cada uno enviado en una línea, que permiten indicar al servidor los tipos de contenidos que acepta el cliente, el lenguaje, los sistemas de codificación, etc.

Vamos a ver sólo alguno de los más importantes. Muchos de los que no veremos será debido a que son parte del complejo sistema de cache que ocuparía por si sólo todo un artículo. A continuación muestro una lista bastante completa de campos de cabecera, en los que señalo aquellos que comentaré:

Accept, Accept-Charset, Accept-Encoding, Accept-Language, Accept-Ranges, Age, **Allow, Authorization**, Cache-Control, **Connection, Content-Encoding, Content-Language, Content-Length**, Content-Location, Content-MD5, Content-Range, **Content-Type, Date**, ETag, **Expect**, Expires, From, **Host**, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Las-Modified, Location, **Max-Forwards**, Pragma, **Proxy-Authenticate, Proxy-Authorization**, Range, **Referer, Retry-After, Server**, TE, Trailer, Transfer-Encoding, Upgrade, **User-Agent**, Vary, Via, Warning, **WWW-Authenticate**.

5.2.1. Accept:

Permite especificar el tipo de contenidos que aceptará el cliente. Consiste en una lista de tipos/subtipos de contenidos. Por tanto, si especificamos ***/*** significa que aceptamos cualquier tipo de contenido.
Por ejemplo:

Accept: text/html, image/png

Especifica al servidor que sólo aceptamos texto HTML e imágenes en formato PNG.
En cambio, si enviamos:

Accept: */*

Indicamos al servidor que aceptamos cualquier tipo de contenidos.
Y si enviamos esto:

Accept: text/html, text/plain, image/*

Indicamos que aceptamos texto HTML, texto plano, y cualquier formato de imágenes.

Podemos especificar, además, prioridades en los formatos. Por ejemplo, si preferimos que las imágenes nos sean enviadas en formato PNG, enviaremos:

Accept: image/png, image/*; q=0.1

Esto significa que la prioridad para cualquier formato de imagen que no sea image/png, es 0.1. El valor de prioridad por defecto es 1, por lo que, al no especificar un valor de prioridad para image/png, se asume que este es 1, el cual es mayor que 0.1. Por tanto, la petición anterior sería equivalente a esta:

Accept: image/png; q=1, image/*; q=0.1

El valor de prioridad (**q**) puede tomar cualquier valor comprendido entre 0 y 1, con hasta 3 decimales, por lo que podemos especificar listas de prioridades bastante complejas. Podemos especificar prioridades de forma mucho más sencilla, simplemente mediante el orden

en que especificamos los tipos aceptados. Por ejemplo:

Accept: text/html, image/png, */*

Especifica al servidor que preferimos siempre texto HTML a cualquier otra cosa, y en segundo lugar, imágenes PNG. En cambio, si no se nos puede enviar ninguna de esas dos cosas, también nos conformamos con cualquier otro tipo de contenido (*/*).

Los tipos básicos están especificados en el **RFC 2046**, y son los siguientes: text, image, audio, video, application. Aunque también se puede hablar de otros tipos más complejos, como el multipart, en los que no voy a entrar en detalle.
Si queréis la lista completa de tipos actualmente registrados, la mantiene el **IANA** (**I**nternet **A**ssigned **N**umbers **A**uthority), y la podéis consultar por FTP en ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/

5.2.2. Accept-Charset:

Especifica al servidor las tablas de caracteres que acepta nuestro cliente. El formato es el mismo que en **Accept**. Por ejemplo:

Accept-Charset: iso-8859-1, unicode-1-1; q=0.8

Indica al servidor que preferimos caracteres según el estándar iso-8859-1, pero que aceptamos también unicode-1-1 (aunque con menor prioridad).
En el número anterior de la revista ya se habló algo sobre estos estándares de juegos de caracteres.

Como ejemplo, en
http://www.htmlhelp.com/reference/charset/ tenéis la tabla de caracteres del estándar iso-8859-1.

** No incluimos “fotos” de la tabla porque ocuparía varias páginas de la revista y en el foro de hack x crack después nos echan bronca por desperdiciar espacio :)

5.2.3. Accept-Encoding:

Especifica al servidor el tipo de codificación de datos que puede comprender nuestro cliente. Para hacer nuestras pruebas lo mejor es que utilicemos sólo la codificación **identity**, es decir, sin ninguna codificación. Así, podremos ver en texto plano todos los datos recibidos, y analizarlos con mayor facilidad. La codificación identity es la que se asume cuando no se indica nada, y deben implementarla todos los servidores. Si aún así queremos especificarlo, enviaremos:

Accept-Encoding: identity

Si queremos experimentar con otras codificaciones, el formato es el mismo que en los casos anteriores. Por ejemplo:

Accept-Encoding: gzip; q=1, compress; q=0.8, identity; q=0.5, *; q=0.1

Con esto indicamos al servidor que preferimos la codificación gzip, seguida de la compress, seguida de la identity y, en último caso, aceptamos cualquier otra codificación existente. Si el servidor no es capaz de enviar una respuesta en ninguna de las codificaciones especificadas, responderá con un **error 406 (Not Acceptable)**.

5.2.4. Accept-Language:

Especifica al servidor el idioma en que preferimos que nos sea enviado el texto. Por supuesto, para que nos sea enviado en un determinado lenguaje, el servidor tiene que tener prevista esta situación y tener traducciones de los contenidos a diversos idiomas.

No sólo se pueden especificar idiomas, si no también subtipos dentro del idioma. Por ejemplo:

Accept-Language: en-gb; q=1, en; q=0.5, *; q=0.1

Indicamos así al servidor que preferimos el idioma inglés de Gran Bretaña (en-gb). Existen otros subtipos, como por ejemplo el en-US, que es inglés de Estados Unidos. Como segunda opción, admitimos cualquier tipo de inglés, aunque no sea en-gb. Por último, con mínima prioridad, aceptamos cualquier otro idioma.

Si no se especifica nada en Accept-Language, el servidor asume que no hay preferencias de idioma, y enviará el que más le convenga.

5.2.5. Authorization:

Si intentamos acceder a una página cuyo contenido requiera la autenticación de un usuario, el servidor nos responderá con un **error 401 (Unauthorized)**. En esta respuesta incluirá además un campo de cabecera **WWW-Authenticate**, en el cual nos dará los datos necesarios para que realicemos la autenticación.

A continuación, tendremos que volver a pedir la página, pero incluyendo los datos de autenticación, lo cual se hace mediante esta línea de cabecera. El sistema de autenticación de HTTP viene especificado en el **RFC 2617**.

5.2.6. Connection:

Permite especificar al servidor si queremos que la conexión se cierre una vez enviada la respuesta:

Connection: close

O si queremos mantenerla abierta para posteriores peticiones:

Connection: Keep-Alive

Por defecto, se considera que se mantiene en el estado **Keep-Alive**, por lo que sólo es necesario indicar explícitamente cuándo se desea cerrar la conexión.

5.2.7. User-Agent:

Esta línea de cabecera permite indicar al servidor información sobre nuestro navegador, como el software utilizado, su versión, el sistema operativo, o incluso algunas compatibilidades. Por ejemplo:

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Linux 2.4.18-14 i686) Opera 6.11 [en]

Todo esto es lo que envía por defecto Opera 6.11 para Linux cada vez que se conecta a una página web. Como vemos, da información no sólo sobre la versión exacta del navegador, si no también sobre el sistema operativo y el microprocesador.



Imagina la cantidad...

Imagina la cantidad de información que obtiene de ti cualquier Servidor Web cuando te conectas a una Web alojada en dicho Servidor :p

Como todas las cosas, esta es un arma de doble filo. Puede utilizarse para, por ejemplo, saber la resolución de tu pantalla y servirte una página Web adaptada a dicha resolución (eso es bueno), pero también puede utilizarse para saber, por ejemplo, la versión exactade tu navegador y servirte una página que "explote" un bug de esa versión del navegador y colarte un troyano (eso ya no es tan bueno ¿verdad?)

5.2.8. Referer:

Si os ha preocupado ver que cada vez que

visitáis una página web se envía información acerca de vuestro sistema y vuestro software, supongo que más os asustará saber que además se envía la última URL que visitasteis. Si enviamos:

Referer: http://www.hackcrack.com/

Indicamos al servidor que estuvimos en esa página, antes de estar en la suya.
Esto puede suponer un gran riesgo de seguridad, ya que se podrían colar datos importantes, como por ejemplo parámetros enviados en una URL (incluso datos de autenticación de otras páginas web).

5.2.9. Allow:

Este campo de cabecera se incluye en las respuestas con código **405 (Method Not Allowed)**, que se envía cuando el cliente intenta utilizar un comando que el servidor no acepta. El servidor informará al cliente mediante este campo de cuáles son los métodos que si que acepta. Por ejemplo:

Allow: GET, HEAD, POST

Informa al cliente de que sólo está permitido utilizar esos tres comandos. El campo Allow también puede ser incluido como respuesta a un **PUT**, indicando así qué métodos se pueden utilizar sobre el nuevo recurso recién creado.

5.2.10. Content-Encoding:

Este campo acompaña a cualquier mensaje en el que se envíen datos, tanto si es una respuesta del servidor a un **GET**, como si son los datos subidos por el cliente en un **PUT** o un **POST**. Si bien el campo **Accept-Encoding** indica qué tipos de codificación aceptamos para la próxima recepción, el campo **Content-Encoding** indica qué tipo de codificación se ha utilizado para un envío. Evidentemente, la codificación indicada en **Content-Encoding** debería ser compatible con alguna de las especificadas previamente por la otra máquina en su campo **Accept-Encoding**.
Por ejemplo:

Content-Encoding: gzip

Indica que se ha utilizado la codificación gzip en los datos enviados a continuación. Para ello, la petición de esos datos tuvo que incluir un campo de cabecera como el siguiente:

Accept-Encoding: gzip; q=1, compress; q=0.5, identity; q=0.1

Es decir, de alguna forma tenía que incluir la codificación gzip entre las admisibles. En este caso concreto, además, era la codificación preferida, así que todos contentos. :-)
Si no se especifica un campo **Content-Encoding**, se asume que no se ha usado codificación, es decir, que se ha usado codificación **identity**.

5.2.11. Content-Language:

La misma relación que existe entre el **Accept-Encoding** y el **Content-Encoding**, existe entre el **Accept-Language** y el **Content-Language**.

5.2.12. Content-Length:

Indica la longitud en bytes de los datos enviados. Por ejemplo:

Content-Length: 1500

Indica que los datos enviados (típicamente una página web) ocupan 1500 Bytes.

5.2.13. Content-Type:

Este campo combina la respuesta a los campos

de cabecera **Accept** y **Accept-Charset**, ya que puede indicar cuál es el tipo de datos enviado, así como el juego de caracteres. Por ejemplo:

Content-Type: text/html; charset=ISO-8859-4

Aunque muchas veces sólo se indicará el tipo/subtipo de los datos. Por ejemplo, en la captura de sesión de ejemplo, vemos que ésta es la respuesta:

Content-Type: text/html

5.2.14. Date:

Como podemos imaginar, indica la fecha y la hora a la que se generó el mensaje en cuestión. Si os queréis complicar la vida, en el punto **3.3.1.** del **RFC 2616** se muestran en detalle los formatos de fechas aceptados, pero supongo que a la mayoría os bastará con un ejemplo, así que no tenéis más que mirar la captura de sesión del ejemplo:

Date: Thu, 09 Oct 2003 14:59:29 GMT

Como vemos, la hora siempre se da con respecto al **GMT** (**G**reenwich **M**ean **T**ime, es decir, la hora en el meridiano 0).

5.2.15. Expect:

Este campo lo utiliza un cliente cuando quiere solicitar al servidor un determinado tipo de acción. Por ejemplo, se usa cuando el cliente espera que el servidor le envíe un **código 100 (Continue)**. Todo esto lo veremos más adelante cuando comentemos las respuestas del servidor. Cuando el servidor no pueda satisfacer las expectativas del cliente, responderá con un **error 417 (Expectation Failed)**.

5.2.16. Max-Forwards:

Este campo de cabecera se utiliza en los comandos **OPTION** y **TRACE** cuando se quiere acceder a una máquina intermedia dentro de la cadena que nos comunica con el servidor final. Como vimos al principio, para llegar a un servidor nuestras peticiones pueden estar pasando por una serie de proxies y routers. La forma de acceder a estos puntos intermedios es mediante esta cabecera.

En este campo se indica como parámetro un número, que es el número de pasos intermedios a dar. Cada vez que pasa por un proxy esta petición, el proxy restará en una unidad este número, y pasará el mensaje al próximo proxy. Cuando el mensaje llegue con valor 0 a uno de los proxies de la cadena, éste ya no realizará más restas, ni retransmitirá el mensaje, si no que lo devolverá con la respuesta pertinente.

Este es un claro ejemplo de modificación de las cabeceras por parte de los proxies. Por ejemplo:

Max-Forwards: 3

Indica que a este mensaje habrá de responder el tercer proxy de la cadena. Si el servidor final estuviese casualmente conectado por menos de 3 proxies, sería éste el que respondería.

5.2.17. Proxy-Authenticate:

Este campo se incluye en la respuesta de un proxy que requiere autenticación. El campo **WWW-Authenticate** requiere autenticación al usuario con el servidor final, y el campo **Proxy-Authenticate** requiere autenticación con uno de los proxies intermedios de la cadena.

El campo **Proxy-Authenticate** se incluye en las respuestas con **código 407 (Proxy Authentication Required)**.

5.2.18. Proxy-Authorization:

Si bien a **WWW-Authenticate** se responde con **Authorization**, a **Proxy-Authenticate** se responde con **Proxy-Authorization**.

5.2.19. Retry-After:

Este campo se puede incluir en las respuestas con **código 503 (Service Unavailable)**, indicándonos típicamente en segundos el tiempo que nos recomienda el servidor que esperemos antes te reintentar la petición. Por ejemplo:

Retry-After: 120

Nos indica que conviene que esperemos al menos 2 minutos antes de reintentar la petición. También se podría indicar una fecha, en lugar de un tiempo en segundos.

5.2.20. Server:

Este es un campo de cabecera de respuesta muy interesante. En él se incluye información acerca del servidor, la cual puede ser útil para alguien que quiera realizar algún tipo de ataque contra el servidor, ya que lo primero que necesitará conocer es el software que utiliza el servidor. Por ejemplo, en la captura de sesión vemos:

Server: Apache/1.3.26 (Unix) PHP/4.2.3 mod_perl/1.26

El servidor nos indica no sólo qué software utiliza, si no también el número de versión completo, el sistema operativo (Unix), así como algunos módulos instalados.

Como vemos, es una información bastante precisa que puede facilitar en gran medida un ataque contra el servidor.

5.2.21. WWW-Authenticate:

Ya he mencionado antes la función de este campo de cabecera. Os recuerdo que en el **RFC 2617** se detalla el mecanismo de autenticación en HTTP.

5.3. Respuestas del Servidor

Ante cada petición realizada por el cliente, el servidor siempre nos dará una respuesta con el siguiente formato:

HTTP/x.x código respuesta

Donde HTTP/x.x es la versión de HTTP, código es un número que identifica el tipo de respuesta, como veremos más adelante, y respuesta es una frase de respuesta aclaratoria del código devuelto.

Por ejemplo, si intentamos acceder a una URL que no existe en un servidor, nos puede responder algo como esto:

HTTP/1.1 404 Sorry, the page you requested was not found.

5.3.0. Códigos de respuesta

Según el primer dígito del código de respuesta, podemos saber si se trata de una respuesta correcta, de error, informativa, etc. Este es el significado de ese primer dígito:

1xx – Los códigos que empiezan por 1 son **informativos**, es decir, la petición ha sido recibida sin problemas, y el servidor se limita a darnos algún tipo de información. Ya veremos más adelante el significado de esto.

2xx – Estas son las respuestas más habituales, ya que son las que devuelven los datos solicitados por una petición recibida y procesada con **éxito**. Por ejemplo, es la que se envía cada vez que recibimos una página web.

3xx – Estas son las respuestas de **redirección**, que indican que para procesar la petición es necesario realizar acciones previas. Ya lo veremos más adelante.

4xx – Supongo que todos estaréis familiarizados con los códigos que empiezan por 4 (sobre todo con el 404 y el 403), ya que son los **errores de cliente**. Es decir, se envían cada vez que el cliente intenta hacer algo que no está permitido o que no tiene sentido para el servidor.

5xx – Estos son los **errores del servidor**, que se dan cuando un cliente envía una petición válida pero, por el motivo que sea, el servidor no puede procesarla.

A continuación, os muestro la lista completa de códigos de respuesta:

100: Continue
101: Switching Protocols

200: OK
201: Created
202: Accepted
203: Non-Authoritative Information
204: No Content
205: Reset Content
206: Partial Content

300: Multiple Choices
301: Moved Permanently
302: Found
303: See Other
304: Not Modified
305: Use Proxy
307: Temporary Redirect

400: Bad Request
401: Unauthorized
402: Payment Required
403: Forbidden
404: Not Found
405: Method Not Allowed
406: Not Acceptable
407: Proxy Authentication Required
408: Request Time-out
409: Conflict
410: Gone
411: Length Required
412: Precondition Failed
413: Request Entity Too Large
414: Request-URI Too Large
415: Unsupported Media Type
416: Requested range not satisfiable
417: Expectation Failed

500: Internal Server Error
501: Not Implemented
502: Bad Gateway
503: Service Unavailable
504: Gateway Time-out
505: HTTP Version not supported

Veremos ahora en detalle el significado de algunas de estas respuestas.

5.3.1. HTTP/1.1 100 Continue:

Cuando el cliente desea enviar datos en un **PUT** o un **POST**, tendrá que pedir permiso previamente al servidor. Para ello, en su petición incluirá el siguiente campo de cabecera:

Expect: 100-Continue

Cuando el servidor reciba la petición con este campo, tendrá que responder bien con:

HTTP/1.1 100 Continue

O bien con:

HTTP/1.1 417 Expectation Failed

En el primer caso, el cliente sabrá que el servidor le da permiso para enviar los datos. En el segundo caso, el servidor le denegará el permiso. La acción por defecto a tomar por el cliente

cuando no recibe ninguna respuesta, es asumir que tiene permiso. Si luego la recepción falla, ya lo notificará el servidor posteriormente.

5.3.2. HTTP/1.1 200 Ok:

Esta es la respuesta más común, ya que es la que se da cada vez que se recibe una página web. En este caso, cuando es una respuesta a un comando **GET**, los datos de la página acompañan a esta respuesta, precedidos por los campos de cabecera necesarios para especificar el tipo de los datos (**Content-Type**), su longitud (**Content-Length**), etc, etc.

Si, en cambio, es respuesta a un comando **HEAD**, todas estas cabeceras se enviarán igual (**Content-Type**, etc), pero no se enviarán los datos HTML de la página.

Si se trata de una respuesta a un comando **POST** el servidor indicará aquí el resultado de la acción llevada a cabo.

Si se trata de una respuesta a un comando **TRACE** contendrá de vuelta el mensaje que envió el cliente al servidor (o proxy).

5.3.3. HTTP/1.1 201 Created:

Ya vimos anteriormente que esta respuesta se utilizaba en los comandos **PUT** y **POST**, cuando se creaba un nuevo recurso como consecuencia de la ejecución del comando.

5.3.4. HTTP/1.1 204 No Content:

Esta respuesta se enviará cuando un comando se haya procesado con éxito, pero no sea necesario enviar ningún tipo de datos.

5.3.5. HTTP/1.1 300 Multiple Choices:

Cuando el servidor encuentra más de una posible localización para el recurso solicitado, indicará así al cliente las alternativas que tiene. Normalmente, el navegador que utilicemos escogerá una de las opciones de forma transparente al usuario.

5.3.6. HTTP/1.1 301 Moved Permanently:

Cuando la localización de un recurso ha cambiado y ya no se encuentra en esa URL, el servidor nos indicará la nueva localización y, normalmente, nuestro navegador pedirá esa nueva URL de forma transparente al usuario.

5.3.7. HTTP/1.1 304 Not Modified:

A pesar de que esta respuesta forma parte del complejo sistema de cache que he comentado, merece la pena mencionarla. Esta respuesta se da cuando solicitamos una página de la cual teníamos una copia en nuestra cache. Si nuestra copia de cache se corresponde con la que hay aún en el servidor, éste no tendrá que retransmitirnos la página entera (con un **HTTP/1.1 200 Ok**), si no que bastará con que nos devuelva esta respuesta para que sepamos que podemos confiar en la copia de cache.
Podemos ver un ejemplo de esta respuesta en la captura de sesión de ejemplo.

5.3.8. HTTP/1.1 400 Bad Request:

La sintaxis de la petición del cliente es incorrecta, y por tanto no la puede procesar el servidor.

5.3.9. HTTP/1.1 401 Unauthorized:

El servidor exige autenticación para que el cliente pueda acceder a ese recurso. Esta respuesta tendrá que ir acompañada de un campo de cabecera **WWW-Authenticate** que incluya los datos necesarios para que el cliente pueda realizar la autenticación.

5.3.10. HTTP/1.1 403 Forbidden:

Seguro que todos os habréis encontrado más de una vez con un error 403 que, sólo superado por el 404, yo creo que es el más habitual. El servidor no permite el acceso a ese recurso, porque el cliente no tiene privilegios para ello. Es lo que ocurre, por ejemplo, cuando se intenta acceder a un directorio cuyos contenidos no quieren ser mostrados por el servidor.

¡Ojo! Que no hay que fiarse. Si realmente no quieren que accedas a esos contenidos, muchas veces tratarán de engañarte, haciendo que la respuesta a esa petición sea un error 404 en lugar de un 403, para que ceses en tu empeño pensando que ni siquiera existe ese recurso ;p

5.3.11. HTTP/1.1 404 Not Found:

¿Quién no conoce el archiconocido 404? Es la respuesta que da el servidor cuando intentas acceder a una URL que no conoce. :-)

5.3.12. HTTP/1.1 405 Method Not Allowed:

Ya mencioné anteriormente que esta respuesta se usa cuando se envía al servidor un comando que éste no acepta por algún motivo. Como ya dije, esta respuesta debe venir acompañada de un campo de cabecera **Allow** para indicarnos los comandos que sí podemos utilizar.

5.3.13. HTTP/1.1 407 Proxy Authentication Required:

Como ya comenté, esta respuesta nos la da un proxy cuando requiere que nos autentiquemos. Para ello, incluirá un campo de cabecera **Proxy-Authenticate**, del mismo modo que ocurre con el campo **WWW-Authenticate** y la respuesta **401**.

5.3.14. HTTP/1.1 414 Request-URI too long:

La longitud de una petición puede llevar a problemas de seguridad, como los conocidos bugs de buffer overflow. Por tanto, tanto el cliente como el servidor tienen que estar preparados para peticiones de cualquier longitud y, en caso de que se supere la longitud máxima que pueden procesar, devolver un error, en lugar de tratar de procesarla con los problemas que eso conllevaría. En la práctica, una gran mayoría de aplicaciones han pecado incluso repetidas veces de no estar preparadas para este tipo de situaciones.

5.3.15. HTTP/1.1 417 Expectation Failed:

Ya hablé sobre esta respuesta, que se da cuando un servidor no puede responder a la petición expuesta por el cliente en un campo **Expect**, por ejemplo cuando se desea que el servidor dé el visto bueno a un comando **PUT** (**Expect: 100-Continue**).

5.3.16. HTTP/1.1 500 Internal Server Error:

Esta es la respuesta genérica de error cuando el servidor no puede procesar una petición válida.

5.3.17. HTTP/1.1 501 Not Implemented:

Esta respuesta se da cuando el servidor no conoce el comando que ha enviado el cliente. A diferencia del **error 405 (Method Not Allowed)**, en el caso del 405 sí que se conoce el comando, pero el servidor no permite su uso.

En cambio, en este caso, el comando es totalmente desconocido para el servidor.

5.3.18. HTTP/1.1 503 Service Unavailable:

El servidor no puede procesar ninguna petición en ese instante, quizá por una sobrecarga, o por encontrarse en mantenimiento. Supuestamente, en un futuro el servidor podría volver a responder a las peticiones, por lo que es conveniente que el servidor incluya un campo de cabecera **Retry-After** que indique al cliente el tiempo que aconseja esperar antes de reintentarlo.

6. Resumen

Por último, vamos a repasar todo este barullo con la captura de sesión que teníamos de ejemplo. Como habréis podido comprobar, es difícil estructurar todo para hacer una explicación sencilla y coherente, ya que todo está relacionado con todo, y a su vez con otros aspectos externos (como el HTML o el MIME). En consecuencia, la documentación de este protocolo es, en mi opinión, bastante caótica, y para redactar este artículo he tenido que estar con una docena de RFCs abiertos simultáneamente para consulta, así como otras referencias.

Espero que haya merecido la pena, y que al menos este artículo os sirva como referencia para hacer un estudio más detallado realizando vuestras propias capturas con un sniffer, y analizando los resultados con esta guía abreviada que he escrito.

Vamos a tratar de comprender ahora esa sesión de ejemplo que pusimos al principio.

Para hacer esta captura, escribí en mi navegador la URL **www.hackxcrack.com**. Por tanto, al no haber indicado un path, mi navegador (Opera) solicitó en primer lugar el raíz ("/") del servidor, de la siguiente forma:

GET / HTTP/1.1
Host: www.hackxcrack.com

Por supuesto, no bastaba con lanzar el comando, si no que además mandó una serie de campos de cabecera.

El primer campo enviado, **User-Agent**, ya sabemos que identifica nuestro navegador que, en este caso, es Opera para Linux:

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; UNIX) Opera 6.11 [en]

A continuación, enviamos una serie de campos para especificar el formato en el que queremos recibir los datos, lo cual hacemos mediante:

Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*
Accept-Charset: windows-1252, utf-8;q=1.0, utf-16;q=1.0, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0

Con esto, indicamos al servidor que preferimos que el texto esté en formato HTML, las imágenes en PNG, JPEG, GIF, o X-XBitMap (en este orden de preferencia) y, por último, que aceptamos cualquier otro tipo de contenido.

Indicamos también que preferimos los juegos de caracteres Windows-1252, utf-8, utf-16. En segundo lugar, también aceptamos iso-8859-1 y, por último, si el servidor no soporta ninguno de esos caracteres, aceptaríamos cualquier otro.

Además, aceptamos una serie de codificaciones, que serían deflate, gzip, x-gzip, e identity (este último no sería necesario indicarlo explícitamente). Al enviar ***;q=0** estamos diciendo que no admitimos otra codificación que no sea una de las propuestas.
Estos campos de cabecera:

Cookie: phpbb2mysql_data=a%3B1%5B%5etc
Cache-Control: no-cache
TE: deflate, gzip, chunked, identity, trailers

Forman parte de los que no he querido comentar por falta de espacio, por lo que vamos a olvidarnos de ellos. Por supuesto, podéis estudiar su significado en el **RFC 2616**.

Por último, el campo:

Connection: Keep-Alive, TE

Indica al servidor que la conexión será persistente, es decir, que después de esta petición pueden venir otras, por lo que no deseamos que cierre la conexión **TCP/IP** después de satisfacer esta petición.

Vamos a ver ahora qué nos respondió el servidor:

Empezó con un código de respuesta 200, por lo que la petición fue procesada correctamente y el servidor nos envía los datos solicitados:

HTTP/1.1 200 OK

A continuación, nos da una serie de datos informativos, como la fecha y hora, los datos del servidor, y otra serie de datos referidos a temas que no he explicado por falta de espacio pero que, de nuevo, podéis estudiar por vuestra cuenta en el **RFC 2616**:

Date: Thu, 09 Oct 2003 14:59:29 GMT
Server: Apache/1.3.26 (Unix) PHP/4.2.3 mod_perl/1.26
Last-Modified: Fri, 23 May 2003 19:52:13 GMT
ETag: "390062-1893-3ece7bed"
Accept-Ranges: bytes
Age: 0

Por último, nos da información acerca de los contenidos (el código HTML de la página servida) que nos envía. En primer lugar, nos dice la longitud en Bytes (6291), y a continuación el tipo de contenidos de que se trata que, en este caso, es texto HTML.

Content-Length: 6291
Content-Type: text/html

Después de esto vendría todo el chorizo de código **HTML** que contiene la página servida.

Dentro de este código HTML se encontraba lo siguiente:

```
<td width="218" height="121" valign="top"><div align="right"><img src="imagenes/LOGOPCPASOAPASO1.jpg" width="144" height="105" /></div></td>
```

Nuestro navegador, al analizar este código, se dio cuenta de que tenía que solicitar al servidor la URL /imagenes/LOGOPCPASOAPASO1.jpg, por lo que hizo una nueva petición:

GET /imagenes/LOGOPCPASOAPASO1.jpg HTTP/1.1
Host: www.hackxcrack.com

Esta vez incluyó como campo de cabecera adicional el **Referer**, diciendo que la última URL que visitamos fue precisamente la que pedimos anteriormente, es decir, www.hackxcrack.com:

Referer: http:

A continuación, envía una serie de campos de cabecera adicionales referidos al sistema de cache, ya que mi navegador ya tenía una copia de esa imagen en su cache:

If-Modified-Since: Thu, 03 Apr 2003 06:06:09 GMT
If-None-Match: "710029-10d4-3e8bcf51"

Con estos datos, el servidor se dio cuenta de que nuestra copia de cache era la misma que tenía el, por lo que no era necesario que volviese a enviarnos la imagen en cuestión, así que nos respondió con un:

HTTP/1.1 304 Not Modified

Tras lo cual, nuestro navegador cogió del disco duro nuestra copia de cache de la imagen y la incluyó en la página que nos tenía que mostrar.

Como vemos, nos resultó conveniente utilizar una conexión persistente, ya que dentro del código HTML enviado por el servidor se incluían nuevas URLs que teníamos que solicitar al servidor, por lo que resultaba mucho más efectivo hacerlo en una única conexión TCP/IP, en lugar de tener que abrir una nueva cada vez que se quisiera solicitar una imagen o cualquier otro contenido incluido en el código HTML de la página.

Autor: PyC (LCo)

SERVIDOR DE HXC MODO DE EMPLEO

- Hack x Crack ha habilitado tres servidores para que puedas realizar las prácticas de hacking.

- Las IPs de los servidores de hacking las encontrarás en EL FORO de la revista (www.hackxcrack.com). Una vez en el foro entra en la zona COMUNICADOS DE HACK X CRACK (arriba del todo) y verás varios comunicados relacionados con los servidores. No ponemos las IP aquí porque es bueno acostumbrarte a entrar en el foro y leer los comunicados. Si hay alguna incidencia o cambio de IP o lo que sea, se comunicará en EL FORO.

- Actualmente tienen el BUG del Code / Decode. La forma de "explotar" este bug la explicamos extensamente en los números 2 y 3. Lo dejaremos así por un tiempo (bastante tiempo ;) Nuestra intención es ir habilitando servidores a medida que os enseñemos distintos tipos de Hack.

- En los Servidores corre el Windows 2000 con el IIS de Servidor Web. No hemos parcheado ningún bug, ni tan siquiera el RPC y por supuesto tampoco hemos instalado ningún Service Pack. Para quien piense que eso es un error (lógico si tenemos en cuenta que el RPC provoca una caída completa del sistema), solo decirte que AZIMUT ha configurado un firewall desde cero que evita el bug del RPC, (bloqueo de los puertos 135 (tcp/udp), 137 (udp), 138 (udp), 445 (tcp), 593 (tcp)). La intención de todo esto es, precisamente, que puedas practicar tanto con el CODE/DECODE como con cualquier otro "bug" que conozcas (y hay cientos!!!). Poco a poco iremos cambiando la configuración en función de la experiencia, la idea es tener los Servidores lo menos parcheados posibles pero mantenerlos operativos las 24 horas del día. Por todo ello y debido a posibles cambios de configuración, no olvides visitar el foro (Zona Comunicados) antes de "penetrar" en nuestros servidores.

- Cada Servidor tiene dos unidades (discos duros duros):
* La unidad c: --> Con 40GB y Raíz del Sistema
* La unidad d: --> Con 40GB
* La unidad e: --> CD-ROM

Nota: Raíz del Servidor, significa que el Windows Advanced Server está instalado en esa unidad (la unidad c:) y concretamente en el directorio por defecto \winnt\ Por lo tanto, la raíz del sistema está en c:\winnt\

- El IIS, Internet Information Server, es el Servidor de páginas Web y tiene su raíz en c:\inetpub (el directorio por defecto)

Nota: Para quien nunca ha tenido instalado el IIS, le será extraño tanto el nombre de esta carpeta (c:\inetpub) como su contenido. Pero bueno, un día de estos os enseñaremos a instalar vuestro propio Servidor Web (IIS) y detallaremos su funcionamiento.

De momento, lo único que hay que saber es que cuando TÚ pongas nuestra IP (la IP de uno de nuestros servidores) en tu navegador (el Internet explorer por ejemplo), lo que estás haciendo realmente es ir al directorio c:\Inetpub\wwwroot\ y leer un archivo llamado default.htm.

Nota: Como curiosidad, te diremos que APACHE es otro Servidor de páginas Web (seguro que has oído hablar de él). Si tuviésemos instalado el apache, cuando pusieses nuestra IP en TU navegador, accederías a un directorio raíz del Apache (donde se hubiese instalado) e intentarías leer una página llamada index.html ... pero... ¿qué te estoy contando?... si has seguido nuestra revista ya dominas de sobras el APACHE ;)

Explicamos esto porque la mayoría, seguro que piensa en un Servidor Web como en algo extraño que no saben ni donde está ni como se accede. Bueno, pues ya sabes dónde se encuentran la mayoría de IIS (en \Inetpub\) y cuál es la página por defecto (\Inetpub\wwwroot\default.htm). Y ahora, piensa un poco… … ¿Cuál es uno de los objetivos de un hacker que quiere decirle al mundo que ha hackeado una Web? Pues está claro, el objetivo es cambiar (o sustituir) el archivo default.html por uno propio donde diga "hola, soy DIOS y he hackeado esta Web" (eso si es un lamer ;)

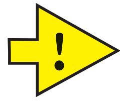
A partir de ese momento, cualquiera que acceda a ese servidor, verá el default.htm modificado para vergüenza del "site" hackeado. Esto es muy genérico pero os dará una idea de cómo funciona esto de hackear Webs ;)

- Cuando accedas a nuestro servidor mediante el CODE / DECODE BUG, crea un directorio con tu nombre (el que mas te guste, no nos des tu DNI) en la unidad d: a ser posible y a partir de ahora utiliza ese directorio para hacer tus prácticas. Ya sabes, subirnos programitas y practicar con ellos :) ... ... ¿cómo? ¿que no sabes crear directorios mediante el CODE/DECODE BUG... repasa los números 2 y tres de Hack x Crack ;p

Puedes crearte tu directorio donde quieras, no es necesario que sea en d:\mellamojuan. Tienes total libertad!!! Una idea es crearlo, por ejemplo, en d:\xxx\system32\default\10019901\mellamojuan (ya irás aprendiendo que cuanto mas oculto mejor :)

Es posiblemente la primera vez que tienes la oportunidad de investigar en un servidor como este sin cometer un delito (nosotros te dejamos y por lo tanto nadie te perseguirá). Aprovecha la oportunidad!!! e investiga mientras dure esta iniciativa (esperemos que muchos años).

- En este momento tenemos mas de 600 carpetas de peña que, como tu, está practicando. Así que haznos caso y crea tu propia carpeta donde trabajar.



MUY IMPORTANTE...

MUY IMPORTANTE!!!!! Por favor, no borres archivos del Servidor si no sabes exactamente lo que estás haciendo ni borres las carpetas de los demás usuarios. Si haces eso, lo único que consigues es que tengamos que reparar el sistema servidor y, mientras tanto, ni tu ni nadie puede disfrutar de él :(Es una tontería intentar "romper" el Servidor, lo hemos puesto para que disfrute todo el mundo sin correr riesgos, para que todo el mundo pueda crearse su carpeta y practicar nuestros ejercicios. En el Servidor no hay ni Warez, ni Programas, ni claves, ni nada de nada que "robar", es un servidor limpio para TI, por lo tanto cuídalo un poquito y montaremos muchos más :)

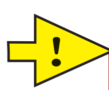
PORT SCANNING

ESCANEANDO ORDENADORES REMOTOS: TIPOS DE SCANEOS

Para utilizar programas/herramientas de "escaneo" como el NMAP necesitamos conocer la forma en que se establecen las conexiones y adentrarnos en su funcionamiento. Este artículo te ilustrará sobre el tema.

1- Avanzando en el análisis de sistemas

Ya hemos comentado la importancia que tiene el análisis previo de un sistema de cara a saltarnos su seguridad, hoy en día existen numerosas técnicas de 'escaneo' de puertos y la mayoría ya han sido implementadas; Un ejemplo claro es NMap del que ya se habló en la revista.



Muy importante

MUY IMPORTANTE: Para cuando leas este artículo, en la Web de PC PASO A PASO (www.hackxcrack.com) podrás descargarte el artículo "Técnicas del Port Scannig y uso del NMAP", que fue publicado en el número 9 de PC PASO A PASO.

Ambos artículos se complementan y nuestra intención es que puedas comprender este texto aunque no comprases en su momento el número 9 de PC PASO A PASO.

Quizás seas de los que piensa que un 'simple escaneo de puertos' no es la mejor forma para entrar en un sistema... aunque sea simplemente uno de los pasos previos evidentemente te equivocas, imaginemos por ejemplo que un usuario aburrido se dedica a jugar con el 'Intesné' escaneando la red y tiene una lista con 250.000 host's con interesante información acerca de su sistema operativo, puertos abiertos, etc.

Ahora imagina que se publica una vulnerabilidad para algún demonio (proceso/programa) de algún sistema operativo como IRIX, FreeBSD o determinada versión de Linux. Nuestro usuario coge su lista y busca coincidencias, ¡acaba de encontrar un centenar de máquinas vulnerables en las que obtener privilegios de administrador! ¿A que no te haría gracia estar en la lista de nuestro amigo?

Por ese motivo resulta interesante conocer mejor el funcionamiento de diferentes técnicas de "port scanning" y de como usar la información obtenida para explotar las debilidades de un sistema así como la forma de protegernos de estos ataques en la medida en que esto sea posible ;-)

2- Algunas nociones fundamentales

Es interesante conocer algunas cosas acerca del protocolo TCP de cara a comprender varias cosas que leerás a continuación y por ello volveremos a mencionar algunos de los conceptos de artículos anteriores.

El protocolo de control de transmisión (TCP) es orientado a conexión y esto implica que se genera un circuito (virtual) entre dos host's cuya comunicación se considera fiable, TCP asegura unas condiciones
óptimas para el circuito establecido y utiliza lo que se denomina "acuse de recibo" para garantizar que los datos lleguen correctamente a su destino. Cuando queremos establecer una

conexión ambas partes deberán estar de acuerdo en participar o dicha conexión no se podrá realizar.

SEGMENTO TCP

La cabecera de un segmento tcp (20 bytes normalmente) es la siguiente:

1 puerto origen			1 puerto destino
2 número de secuencia			
3 número de secuencia ack			
4 long. cabecera	10 reservado	5 flags	6 ventana
7 checksum			8 puntero urgente
9 opciones			
11 DATOS			

Explicaré muy brevemente los campos ;) **

**** Seguramente no comprenderás muchos de los conceptos que a continuación se detallarán, no te preocupes demasiado, la idea es que empiecen a "sonarte" un montón de nombres "raros" y tener una visión global del asunto. Para comprender a la perfección una conexión TCP/IP necesitarías programar sockets en lenguaje C, algo que no tardaremos mucho en publicar... por el momento no te agobies e intenta simplemente tener una visión general. Para practicar todo lo que a partir de ahora se explicará puedes emplear el NMAP.**

1. Puerto origen y destino, bastante explícito :P

2. Número de secuencia, al intentar establecer una conexión los host's que intervienen en el proceso eligen un número aleatorio para empezar a contabilizar bytes que viajarán en los segmentos de datos de la conexión. Los sucesivos segmentos que se envíen llevarán como número de secuencia el número aleatorio elegido al principio más el número de bytes enviados hasta ese momento.

3. El número de secuencia ack (o acknowledge number) es lo que permite validar los segmentos que van llegando a un host, con este fin se coloca en el campo el número de secuencia del segmento incrementado en 1, dicho byte espera ser recibido en el siguiente envío. **

**** Los puntos 2 y 3 es, para que nos entendamos, la forma que tiene el protocolo TCP de no perder ningún paquete. Si estamos enviando a un compañero un archivo Word de un par de megas, este se corta en pequeños trocitos y el receptor debe recibirlos todos, no puede perderse un solo paquete. Vale, ya se que explicarlo así es muy poco técnico, pero quiero que se entienda.**

4. La longitud de la cabecera en múltiplos de 32 bits. **

**** Hemos dicho que el archivo Word es cortado en paquetitos ¿verdad? Bueno, pues cada paquetito es como una carta. La CABECERA de un paquete TCP es como el sobre de una carta, es donde figuran los datos del remitente (el que envía la carta) y del destinatario (el que debe recibir la carta). Dentro del sobre ¿qué encontramos?, pues lo importante, el contenido, los DATOS, lo que quieren comunicarnos, que nos ha tocado la lotería y cosas de ese tipo ;p Pues bien, un paquete TCP es exactamente igual que una carta, tenemos una CABECERA (el sobre) y unos DATOS (en este caso un cachito del archivo WORD que estamos enviando).**

5. Ahora lo que nos interesa especialmente, el campo 'flags' (banderas):

Tenemos varios flags para disfrute personal y son URG|ACK|PSH|RST|SYN|FIN ahora te explico algo más acerca de ellos, como verás no tienen ningún misterio :)

URG, indica que el segmento transporta datos urgentes.

ACK, indica que el número de secuencia ack de la cabecera es válido.

PSH, fuerza el envío inmediato de los datos recibidos al nivel de aplicación, que serían las aplicaciones finales a nivel de usuario.

RST, indica que la comunicación debe reiniciarse.

SYN, durante la conexión indica el proceso de sincronización.

FIN, indica que el host desea finalizar la conexión.

6. El tamaño de la ventana es el número de bytes que esperan ser recibidos sin necesidad de ser validados por parte de un host y puede variar durante la conexión activa.

7. La integridad de los datos y la cabecera tcp se pueden verificar mediante el checksum o suma de verificación.

8. El puntero urgente (que se utiliza junto al flag URG) indica el número de secuencia del último byte que se considera parte de los datos fuera de banda (urgentes).

9.Opciones TCP como el tamaño máximo de segmento, el 'window scale' y otras que de momento no vamos a ver aquí.

10,11. Un campo reservado y de datos respectivamente.

¿COMO SE ESTABLECE UNA CONEXIÓN ORDENADA TCP?

El proceso se denomina three-way handshake (o saludo en tres fases)

```
    HOST A                              HOST B

 1)   |------------------------------>[SYN]
      [SYN][ACK]<-------------------------|  2)
 3)   |------------------------------>[ACK]
```

Es posible que no comprendas el esquema pero tranquilidad que ahora te lo explico paso a paso ;-) seguro que lo entenderás mejor:

El proceso para recibir una petición de conexión implica estar preparado para llevar a cabo una serie de llamadas a funciones como lo son bind (), listen () o connect () de las que hablaremos en el texto de forma genérica.

1) Cuando el host A desea establecer una conexión mediante un circuito TCP envía un segmento al host B, el segmento llevará el flag SYN levantado indicando el proceso de sincronización y no suele llevar ningún tipo de datos salvo la cabecera IP y TCP así como las posibles opciones TCP si las hay. El host A genera una llamada a connect () para el propósito.

2) Ahora el host B enviará la validación para el segmento anterior y activará el flag ACK y en el número de secuencia ack colocará el número de secuencia del segmento recibido + 1, como el proceso de sincronización no ha terminado el flag SYN sigue levantado.

3) El host A sabe que el host B ha validado la petición al recibir el segmento pero ahora el host B está esperando a que se valide su segmento. El host A envía un segmento colocando el valor oportuno en el número de secuencia ack y activando el flag ACK, el flag SYN no viaja levantado esta vez. En este punto la conexión se ha completado con éxito.

¿COMO SE FINALIZA UNA CONEXIÓN ORDENADA TCP?

```
       Host A (cliente)     Host B (servidor)

1)          |------------------>[FIN]
            [ACK]<------------------|  2)
            [FIN]<------------------|  3)
4)          |------------------>[ACK]
```

Ahora te explico que significa eso, no te impacientes, ya sabes que aquí siempre lo explicamos todo para que no se nos escape ningún detalle :)

1) El host A genera una llamada a close() para finalizar la conexión y se envía un segmento con el flag FIN levantado.

2) El host B valida cualquier información previa enviando un segmento con el flag ACK levantado.

3) Para que el host A pueda cerrar la conexión el host B debe enviar un segmento con el flag FIN activado.
4) Se valida el segmento anterior y la conexión se cierra con éxito.

**** Todo esto ocurre de forma transparente al usuario cada vez que establecemos una conexión con un pc remoto (por ejemplo cuando hacemos un simple PING o nos conectamos a una Web con el navegador). De**

hecho suceden muchas más cosas, pero nos conformamos con que captes el concepto general de que establecer una conexión es un proceso de 3 pasos y cerrarla es un proceso de cuatro pasos y que los "flags" (BANDERAS) tienen mucho que decir en este proceso :)

3- Las técnicas de "port scanning"

Existen muchas técnicas para descubrir qué puertos tiene abiertos un host y determinar que servicios están corriendo e incluso bajo que privilegios, pero lo más importante es hacerlo de forma sigilosa sin que la actividad quede registrada en los 'logs' de nuestra víctima y evitar que los sistemas de detección de intrusos (IDS) nos apunten como origen del escaneo. Te aseguro que a muchos usuarios no les hará gracia que les toques los... puertos :P

Vamos a ver detalladamente varias de las técnicas de escaneo ya conocidas, algunas son una maravilla :) y otras no tanto pero en cualquier caso debes conocerlas para poder adaptar mejor el escaneo a tus necesidades jeje. Ya empezaste a leer acerca de ellas cuando tratamos en la revista el NMap pero ahora veremos con más detalle esas y otras técnicas como el 'Idle Scan' :)

- TCP CONNECT (), mediante esta técnica no necesitarás ningún tipo de privilegio especial como sucederá con otros tipos de escaneo como TCP SYN, y además es una técnica muy rápida puesto que podemos efectuar varias conexiones en paralelo. Su funcionamiento (la mayoría ya lo habéis deducido) es el siguiente: intentamos establecer una conexión con un puerto determinado si el puerto está abierto la llamada a connect () tiene éxito y se nos retornará el valor oportuno en caso contrario la llamada fallará. El problema reside en que los intentos de conexión se registrarán automáticamente en el sistema y en principio no nos interesa en absoluto dejar huellas de ningún tipo.

- TCP SYN (), hace un momento te acabo de explicar como se establece una conexión completa, mandamos un paquete con el flag SYN tal y como se haría normalmente al intentar establecer una conexión y ahora la máquina remota nos responde amablemente con SYN|ACK hasta aquí va todo perfecto pero ahora nosotros para llevar la contraria NO vamos a responder con ACK para establecer la conexión, en lugar de eso la abortaremos mediante un paquete con RST, ¿y eso por qué? Te preguntarás, la respuesta es muy sencilla, si ya sabemos que hay alguien al otro lado (tras recibir SYN|ACK) para que seguir intentando conectar ¿? así evitamos establecer la conexión completa y es poco probable que nuestro escaneo quede registrado que es de lo que se trata ;) este escaneo probablemente pasará desapercibido pero recuerda, eso sí, que se necesitan privilegios de administrador para usar esta técnica y que el sistema pueda montar ese tipo de paquetes, a este tipo de escaneo se le conoce también como escaneo "medio abierto".

- TCP FIN, aunque el escaneo TCP SYN no suele dejar rastro en los logs del sistema y su detección puede ser algo complicada hay algunos entornos sensibles al escaneo SYN como aquellos en los que hay filtros de paquetes o "firewalls" y por otro lado existen utilidades que registrarán los intentos de conexión SYN. Para estos casos puede resultar de interés echar mano al escaneo sigiloso FIN (o stealth scan), se fundamenta en el hecho de que los puertos abiertos ignoran los paquetes recibidos con el flag FIN (vacío) y los cerrados responderán con RST. Como se dijo en su momento al presentar Nmap los sistemas de Micro$oft entre otros no son susceptibles a este tipo de ataques pues no siguen las pautas establecidas, quien lo diría xD ¿verdad?

- ACK scan, para esos "ambientes hostiles" con la presencia de cortafuegos puede interesar "nmapear" las reglas de filtrado. Mediante esta técnica podemos enviar un paquete con el flag ACK con los números de secuencia y ack de forma incorrecta o aleatoria de manera que recibamos un paquete de control ("ICMP unreachable", inalcanzable) o no se reciba respuesta, en tal caso el puerto se encuentra filtrado y en caso contrario la conexión será reiniciada y las conexiones al puerto no estarían filtradas. Como habrás deducido, no puedes usar este tipo de escaneo para averiguar los puertos abiertos. Si deseas además encontrar los puertos abiertos y además clasifica los puertos según se encuentren filtrados o no, existe otra técnica conocida como 'Window Scan' que además aprovecha anomalías el tamaño de la ventana TCP por parte de varios sistemas operativos para determinar si el puerto está abierto.

- UDP scan (User Datagram Protocol scan), antes de nada aclarar que se trata de un escaneo mediante un protocolo "no orientado a conexión" de manera que no existe

un circuito virtual entre host's ni tampoco garantía de entrega alguna de los paquetes enviados. Se utiliza cuando queremos averiguar que puertos udp (un puerto puede ser tcp y udp al mismo tiempo pues difieren en el tipo de protocolo pese a tener el mismo número identificador) están abiertos, si tras mandar el paquete se recibe un mensaje de control informando del error el puerto está cerrado y en caso contrario, no recibir nada, se encuentra abierto. Esta técnica es muy poco precisa ya que si por ejemplo se filtran las conexiones udp aparecerán como abiertos puertos que están bloqueados. Además el número de conexiones udp suele estar limitado por los sistemas de manera que el escaneo puede ser bastante lento, y he dicho que suele limitarse porque existe cierto sistema operativo cuyo nombre no recuerdo ;) que no atiende a las especificaciones establecidas por los RFC's de manera que el escaneo irá mucho más rápido en los sistemas de la compañía Micro$oft. Por el mismo motivo el escaneo denominado 'Null Scan' o escaneo nulo que se basa en no levantar ninguna bandera tampoco funcionará al usarlo contra una máquina Window$:(

- Xmas Scan: muy similar al escaneo nulo esta técnica envía un paquete con todas las banderas levantadas. Si el puerto está abierto debe ignorar el paquete.

- Idle scan, esta es para mí una de las técnicas más interesantes y que realmente interesa poder usar siempre que se pueda ya que es altamente sigilosa y no dejamos ni rastro de nuestra IP :-) esto es así debido básicamente a que no será el origen del escaneo... o mejor dicho no para la máquina víctima de nuestro Idle scan jeje. Con lo leído hasta ahora y para tu total comprensión del Idle Scan lo único nuevo que deberías saber es que los paquetes que se envían llevan un 'identificador de fragmento' que se suele incrementar cada vez que se envía uno. Esta técnica de escaneo utiliza host's intermedios o los llamados "zombies" para escanear un objetivo, ahora te explico la forma de hacerlo:

Elegir un host zombie para determinar su identificación IP (IPid), ahora enviamos un paquete para probar un puerto en nuestra víctima desde el host zombie (ya deberías saber que podemos falsear la dirección IP tal y como se comentó en el artículo del NMap, pero nos interesa obtener los resultados ahora). El resultado puede ser que el puerto este abierto o cerrado (--no me digas) en el primer caso NOSOTROS enviamos un paquete falso como si fuésemos el host ZOMBIE de manera que la respuesta del host VICTIMA irá para el host tal y como muestra el ejemplo, supondremos que el IPid del zombie es ahora 100 e ignoraremos a que puerto se dirige junto con otros datos irrelevantes para hacer el ejemplo un poco más inteligible:

Lo primero es obtener el IPid del zombie escogido (100),

```
NOSOTROS                        ZOMBIE
  (Ipid?) -------- SYN|ACK ---->
          <-------  RST -------- (IPid 100)
```

Tras la primera consulta procedemos,

```
NOSOTROS -------- SYN -----> VICTIMA
                              | ^
ZOMBIE   <------ SYN|ACK ------. |
|                               |
|---------------- RST -----------. (IPid +1 = 101)
```

Chequeamos el IPid para ver si el puerto se encuentra abierto,

```
NOSOTROS                            ZOMBIE
 (Ipid?) -------- SYN|ACK ---->
        <---------  RST ------ (IPid +1 = 102)
```

¿Qué ha pasado arriba?

Pues lo que ha ocurrido es que ahora el IPid del zombie se ha incrementado en 1 tras rebotar al zombie nuestra petición de conexión. Ahora tu que eres una persona muy atenta observas de nuevo el IPid de nuestro amigo el zombie xD y observas que efectivamente vale 100 + 2 = 102 y eso significa que el puerto está abierto :-) pero como no siempre tendremos esa suerte ahora veremos que ocurre si el puerto está cerrado, tomaremos los mismos datos previos pero con esa diferencia respecto al estado del puerto:

```
NOSOTROS -------- SYN -----> VICTIMA
                              | ^
ZOMBIE  <-------- RST ---------. |
|                               |
|---------------- RST -----------.
```

¿Y ahora que ha ocurrido?

Pues muy sencillo :-), al rebotar al zombie la petición de conexión la víctima (a la que también se le puede llamar host remoto :P) nos dice "¿SYN@#~?¿eso qué es?" Como le suena raro y curiosamente el puerto está cerrado nos reinicia la conexión mediante RST. Ahora miramos de nuevo el IPid de nuestro zombie y nos damos cuenta de que solo ha incrementado en una unidad desde la última vez por lo que el puerto estaba cerrado. Si te fijas la IP que recibe la víctima es siempre la de la máquina zombie ;-), muy bien pues ahora imagina que nuestra víctima tiene habilitado un cortafuegos que tiene "permiso" para dejar pasar los paquetes cuya IP coincida con la de nuestro zombie (sucede por ejemplo en las relaciones de confianza), ¡acabas de saltarte el cortafuegos por la cara!. Espero que hayas comprendido bien el funcionamiento de esta potente técnica de escaneo que como ves es muy efectiva. El principal problema es encontrar host's que tengan números IPid predecibles como serían las dedicadas a impresión puesto que tienen poca actividad o tráfico para llevar a buen término esta técnica.

Hasta aquí hemos visto con detalle los tipos de escaneo más importantes y más utilizados. Te recuerdo que explicamos en un artículo anterior la herramienta que aplica estas y otras técnicas de "port scanning" que fue NMap, permíteme que te sugiera dicha utilidad si no te apetece programar tu propio escáner de puertos ahora que ya sabes como funciona el asunto :)

4- UN PASO MÁS: DETECCIÓN REMOTA DE SO's

Ni que decir tiene la importancia que tiene saber a que sistema operativo nos enfrentamos ya que resulta fundamental de cara al análisis y la penetración de un sistema así como la explotación remota debido a fallas conocidas (e incluso desconocidas :P) en alguno de los servicios, etc. Existen varias formas y utilidades que nos permitirán obtener una huella digital (o fingerprint) de nuestra víctima que nos sirva para discriminar un sistema operativo del resto, ahora veremos algunas.

PRIMER CONTACTO

Si ya has realizado conexiones directamente desde la consola ya sea para mirar el correo, usar el FTP y/o cualquier otra cosa que se pueda hacer mediante telnet te habrás fijado de que en muchos casos se nos está dando una valiosa información de forma totalmente gratuita por parte del demonio que corre en ese puerto referente a ellos mismos y en muchos casos la plataforma sobre la que corren :) pero no siempre lo tendremos tan fácil por lo que ahora veremos otras técnicas bastante más "sofisticadas" jeje.

```
linux $ ftp ftp.netscape.com
Connected to ftp.gftp.netscape.com.
220-15
220 ftpnscp.newaol.com FTP server (SunOS 5.8) ready.
Name (ftp.netscape.com:linux): anonymous
500 'AUTH SSL': command not understood.
SSL not available
331 Guest login ok, send your complete e-mail address as password.
Password:
230-The response `' is not valid.
230-Next time please use your e-mail address as password.
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> syst
215 UNIX Type: L8 Version: SUNOS
ftp>

linux $ telnet 131.215.48.89 21
Trying 131.215.48.89...
...
220 secant FTP server (Version wu-2.6.1(2) Thu Nov 29 15:02:58
PST 2001) ready.
ftp> syst
215 UNIX Type: L8

linux $ telnet 216.127.72.117
...
Red Hat Linux release 7.2 (Enigma)
Kernel 2.4.9-34 on an i686
login: ^D
```

ANALIZANDO LA PILA TCP/IP

A este método se le suele llamar "Stack FingerPrinting" y permite obtener conclusiones analizando la pila de protocolos TCP/IP.

Algunas formas de discriminar según el sistema operativo conocidas se basan en:

- Flag de fragmentación, en casos concretos se establece por parte de ciertos SO's el flag DF (deshabilitar la fragmentación) de la cabecera IP.

- Números de secuencia, se intentan encontrar patrones en los números iniciales escogidos al responder a una solicitud de conexión como por ejemplo incrementar un valor X cada cierto tiempo o elegir valores totalmente aleatorios, devolver el mismo número enviado, etc.

- Limitación ICMP, el sistema operativo Linux siguiendo las normas y recomendaciones establecidas limita el número de

mensajes del tipo ICMP unreachable (inalcanzable) a un máximo de 20 por segundo pero no todos los sistemas operativos siguen las reglas establecidas tal y como he comentado antes ;)

- Flag no determinado, enviamos SYN y uno de los 6 bits reservados para ver si se mantienen estos bits levantados en la respuesta ya que algunos sistemas cerraran la conexión tras considerar el error, y como sobre gustos no hay nada escrito... :P

- Tipo de servicio, se puede comprobar uno de los campos del mensaje de control de errores ICMP de tipo "unreachable" para ver si su valor es 0 u otro ya que lo normal es que su valor sea 0 pese haber sistemas, como es el caso de Linux, en que se devuelve un 192 decimal (0xC0).

- Opciones TCP, se pueden observar los valores devueltos, su orden o si son soportadas o no ciertas opciones como lo son el window scale, o el MSS (tamaño máximo de segmento).

- Fragmentos solapados, dependiendo de la implementación del sistema para el tratamiento de los fragmentos solapados, el nuevo fragmento sobrescribe o es sobrescrito por el fragmento anterior, hay que utilizar todos los fragmentos para reconstruir el paquete original completo.

Estos son algunos de los métodos que existen y que se suelen utilizar para determinar con MUCHA precisión que SO corre en nuestra víctima pero no son los únicos y posiblemente aparecerán algunos otros más por lo que no es el objetivo de este artículo profundizar en todos ellos ahora, quizás más adelante en próximos artículos ;-)

Por supuesto existen potentes utilidades como QUESO (Qué Sistema Operativo) que puedes obtener de http://www.apostols.org con el objetivo de detectar el SO de un host remotamente. QUESO utiliza varios paquetes con ACK=0 y un número de secuencia aleatorio de la siguiente forma:

1 SYN
2 SYN|ACK
6 PSH
3 FIN
4 SYN|FIN
5 FIN|ACK
7 SYN|otros flags no estandarizados.

QUESO es un programa bien diseñado mediante un fichero aparte que contiene las respuestas que se esperan según el tipo de paquete enviado y que se utilizada para contrastar los resultados obtenidos para afirmar con bastante seguridad que SO corre en nuestro objetivo, en fin una maravilla que ya deberías tener :)

5- PortSentry

Mediante esta interesante utilidad que te puedes descargar de su sitio web oficial en http://www.psionic.com (pero suele venir el paquete correspondiente en muchas de las distribuciones de Linux) capaz de detectar y responder en tiempo real a un escaneo de puertos contra una máquina ¡incluso puede detectar escaneos ocultos! aunque no en todos los casos evidentemente ;-), PortSentry se pone a la escucha en los puertos no utilizados que se indican en /etc/portsentry.conf (por lo general) y guarda las direcciones IP origen del escaneo guardándolas a continuación en /etc/host.deny entonces se puede dejar en un host inválido como 999.999.999.999 mediante TCPWrappers o utilidades similares. Evidentemente necesitarás algo más para proteger tu sistema como un cortafuegos y una política de seguridad adecuada... :P

La línea de comandos básica para PortSentry teniendo en cuenta que estamos bajo Linux como super usuario es:

root # portsentry –stcp, detectar escaneos ocultos TCP.

root # portsentry –atcp, como el anterior pero en modo avanzado.

root # portsentry –tcp, detección básica TCP con atadura al puerto (con binding o ligado al puerto).

root # portsentry –udp, detección básica UDP con atadura al puerto.

root # portsentry –sudp, puede detectar escaneos UDP ocultos.

root # portsentry –audp, como -sudp pero en modo avanzado.

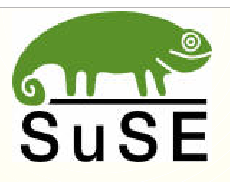
Puede que hablemos de esta y otras utilidades relacionadas en un futuro pero por ahora esto es más que suficiente :) para empezar.

6- Medidas de seguridad básicas

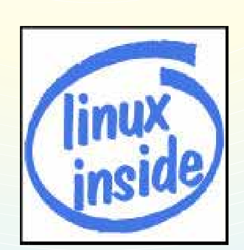
Como has podido observar existen numerosas formas de escanear una red y obtener información sensible acerca de la misma por lo que se recomienda instalar sistemas IDS que registren cualquier actividad sospechosa así como un buen cortafuegos (con una buena política restrictiva) y disponer de un buen 'syslog'. Además lo ideal sería guardar todos nuestros "logs" en un servidor remoto junto con sumas de verificación u otros sistemas que nos aseguren que no se han modificado nuestros registros. Existen muchas aplicaciones para proteger nuestro sistema del exterior pero ninguna medida es suficiente como para proteger al 100% nuestro sistema, eso es lo único que es realmente seguro si tu computadora está conectada a la red de redes :P y lo seguiremos demostrando en los siguientes artículos ;-)

EL GANADOR DEL SORTEO DE UN SUSE LINUX 8.2 DEL MES DE SEPTIEMBRE ES:
JOSE RUIZ BALLESTER
MADRID
SEGUIR LLAMANDO, EL PROXIMO PODRIA SER PARA TI (PAG 66)





Incluye 7 CD's y 1 DVD
Manual de Instalación.
Manual de Administracion





NÚMERO1:

-CREA TU PRIMER TROYANO
INDETECTABLE POR LOS ANTIVIRUS.
-FLASHFXP: SIN LÍMITE DE VELOCIDAD.
-FTP SIN SECRETOS: PASVMODE.
-PORT MODE/PASV MODE Y LOS FIREWALL: LA UTILIDAD DE LO APRENDIDO.
-TCP-IP:INICIACIÓN (PARTE 1).
-EL MEJOR GRUPO DE SERVIDORES FTP DE HABLA HISPANA.
-EDONKEY 2000 Y SPANISHARE.
-LA FLECHA ÁCIDA.



NÚMERO 2:

-CODE/DECODE BUG: INTRODUCCIÓN.
-CODE/DECODE BUG: LOCALIZACIÓN DEL OBJETIVO.
-CODE/DECODE BUG: LÍNEA DE COMANDOS.
-CODE/DECODE BUG: SUBIENDO ARCHIVOS AL SERVIDOR REMOTO.
-OCULTACIÓN DE IP: PRIMEROS PASOS.
-LA FLECHA ÁCIDA: LA SS DIGITAL.
AZNAR AL FRENTE DE LA SS DEL SIGLO XXI.



NÚMERO 3:

-PROXY: OCULTANDO NUESTRA IP.
ASUMIENDO CONCEPTOS.
-PROXY: OCULTANDO NUESTRA IP.
ENCADENANDO PROXIES.
-PROXY: OCULTANDO NUESTRA IP.
OCULTANDO TODOS NUESTROS PROGRAMAS TRAS LAS CADENAS DE PROXIES.
-EL SERVIDOR DE HACKXCRACK: CONFIGURACIÓN Y MODO DE EMPLEO.
-SALA DE PRACTICAS: EXPLICACIÓN.
-PRÁCTICA 1ª: SUBIENDO UN ARCHIVO A NUESTRO SERVIDOR.
-PRÁCTICA 2ª: MONTANDO UN DUMP CON EL SERV-U.
PRÁCTICA 3ª: CODE/DECODE BUG. LÍNEA DE COMANDOS.
-PREGUNTAS Y DUDAS.



NÚMERO 4:

- CREA TU SEGUNDO TROYANO, INDETECTABLE E INMUNE A LOS ANTIVIRUS.
CONOCIENDO EL RADMIN.
GESTIONANDO UNA SALA DE ORDENADORES.
OCULTANDO EL RADMIN.
INSTALANDO EL RADMIN EN EQUIPOS REMOTOS.
- OCULTACIÓN DE IP POR NOMBRE DE DOMINIO.
- CREA LETRAS DE IMPACTO PARA TUS DOCUMENTOS (LETRAS DE FUEGO).
- CONSIGUE UNA IP FIJA.



NÚMERO 5:

-HACK-OPINION: LA PIRATERÍA EN INTERNET.
-ROOTKITS: LA PESADILLA DE CUALQUIER ADMINISTRADOR.
-ROOTKITS: EL SR. NTROOT.
-WAREZ: APPZ, GAMEZ, MP3Z, DIVX, FTPZ, 0-DAY.
-APRENDIENDO A COMPILAR PROGRAMAS. COMPILA TU PROPIO NETCAT.
-BUGS, ERRORES Y OTRAS FORMA DE JOD...
-NETBIOS: ESTUDIO Y PENETRACIÓN DE SISTEMAS.
-ASESINADOS POR LA LSSI.
-LISTADO DE ORDENES PARA NETBIOS.
-HACK-OPINION: PAGOS POR INTERNET SEGUROS YÁ.



NÚMERO 6:

- PASA TUS PELICULAS A DIVX (STREAMING)
- PASA TUS PELICULAS A DIVX II (CODEC DIVX)
- PUERTOS & SERVICIOS
- eMule: EL NUEVO REY DEL P2P
- NUEVA SECCION: PROGRAMACION DESDE 0
- CURSO DE VISUAL BASIC
- IPHXC: EL TERCER TROYANO DE HXC
- TENDENCIAS ACTUALES EN CODIGO MALICIOSO
- OCULTACION DE FICHEROS. METODO STREAM (ads)
- TRASTEANDO CON EL HARDWARE DE UNA LAN



NÚMERO 7:
- PROTOCOLOS: POP3
- PASA TUS PELICULAS A DIVX III (EL AUDIO)
- PASA TUS PELICULAS A DIVX IV (MULTIPLEXADO)
- CURSO DE VISUAL BASIC: LA CALCULADORA
- IPHXC: EL TERCER TROYANO DE HXC II
- APACHE: UN SERVIDOR WEB EN NUESTRO PC
- CCPROXY: IV TROYANO DE PC PASO A PASO
- TRASTEANDO CON EL HARDWARE DE UNA LAN



NÚMERO 8:

- CURSO DE LINUX
- APACHE: COMPARTE ARCHIVOS
- REVERSE SHELL
- CURSO DE VISUAL BASIC: MAS CALCULADORA
- PROTOCOLOS Y SU SEGURIDAD: SMTP



NÚMERO 9:

- CURSO DE LINUX (Sistema de archivos)
- APACHE: COMPARTE ARCHIVOS MEDIANTE WEB.
- CURSO DE VISUAL BASIC: MI 1ª DLL \ ACCESO A DATOS
- PORT SCANING: NMAP
- SERIE RAW: IRC



NÚMERO 10:

- CURSO DE LINUX (Gestión de usuários)
- APACHE + MySQL + PHP=Trio de Ases
- CURSO DE VISUAL BASIC: ACCESO A DATOS(II)
- XML: El futuro de la trasferencia de datos
- SERIE RAW: DCC



NÚMERO 11:
- Curso de linux: programacion
- Visual Basic: IIS bug exploit
- Apache como proxy
- Serie Raw: FTP
- Validacion XML: DTD
- Historia: Lady Augusta Ada Byron



NÚMERO 12:
- Curso de linux: programacion C.
- Visual Basic: IIS bug exploit. Nuestro primer Scanner.
- APACHE: Configuralo de forma segura.
- Serie Raw: FTP(II)
- VALIDACION XML: DTD (II)

CONSIGUE LOS NUMEROS ATRASADOS EN: WWW.HACKXCRACK.COM